

**“Software Development (SW) Pathway (Java)”**

# **Mthree Training**

---

**Anjali Chauhan**

16th September to 13th November

## Index

1. [Day1 \[16th Sep 2024\] - SDLC and Agile](#)
2. [Day2 \[17th Sep 2024\] - Git and Github](#)
3. [Day3 \[18th Sep 2024\] - Java Basic and Bank Application Group Activity](#)
4. [Day4 \[19th Sep 2024\] - Java Continue and Activity](#)
5. [Day5 \[20th Sep 2024\] - Caching, Collection and Threads Concepts](#)
6. [Day 6 \[23rd Sep 2024\] - Concepts of Caching](#)
7. [Day 7 \[24th Sep 2024\] - Caching Continue and To Develop a Minor Project](#)
8. [Day 8 \[25th Sep 2024\] - Data Structures : Array and Sorting](#)
9. [Day 9 \[26th Sep 2024\] - Arrays, SQL Basics and Database Connectivity](#)
10. [Day 10 \[27th Sep 2024\] - Spring Concepts and Group Project](#)
11. [Day 11 \[30th Sep 2024\] - Maven Project using Database & Data Structures Concepts](#)
12. [Day 12 \[1st Oct 2024\] - Sorting and Working on LMS and Hackerearth](#)
13. [Day 13 \[3rd Oct 2024\] - Dynamic Programming, Lambda and Practice](#)
14. [Day 14 \[4th Oct 2024\] - Strings, Double Hashing and Practice](#)
15. [Day 15 \[7th Oct 2024\] - SQL Commands](#)
16. [Day 16 \[8th Oct 2024\] - SQL Concepts](#)
17. [Day 17 \[9th Oct 2024\] - DBMS Concepts](#)
18. [Day 18 \[10th Oct 2024\] - Database Concepts](#)
19. [Day 19 \[11th Oct 2024\] - Relational Database Design Concepts](#)
20. [Day 20 \[14th Oct 2024\] - Concepts of Spring Boot](#)
21. [Day 21 \[15th Oct 2024\] - Spring Boot Applications](#)
22. [Day 22 \[16th Oct 2024\] - SQL Views and Spring Boot Applications](#)
23. [Day 23 \[17th Oct 2024\] - Spring Boot Continues...](#)
24. [Day 24 \[18th Oct 2024\] - SQL Practice Questions](#)
25. [Day 25 \[21st Oct 2024\] - Spring Boot Application and Project Discussion](#)
26. [Day 26 \[22nd Oct 2024\] - Spring Boot Concepts](#)
27. [Day 27 \[23rd Oct 2024\] - Spring Continues...](#)
28. [Day 28 \[24th Oct 2024\] - Working with Nodejs and Angular](#)
29. [Day 29 \[25th Oct 2024\] - Angular Continues...](#)
30. [Day 30 \[28th Oct 2024\] - Angular Applications](#)

- 
- 31. [Day 31 \[29th Oct 2024\] - Shopping List Application and Project](#)
  - 32. Day 31 [30th Oct 2024] - LMS and Project
  - 33. Day 32 [4th Nov 2024] - Working on Project
  - 34. Day 33 [5th Nov 2024] - Working on Project
  - 35. Day 34 [6th Nov 2024] - Working on Project
  - 36. Day 35 [7th Nov 2024] - Working on Project
  - 37. Day 36 [8th Nov 2024] - Working on Project
  - 38. Day 37 [11th Nov 2024] - Working on Project
  - 39. Day 38 [12th Nov 2024] - Working on Project
  - 40. Day 39 [13th Nov 2024] - Final Submission of Project

## Day 1 - SDLC and Agile

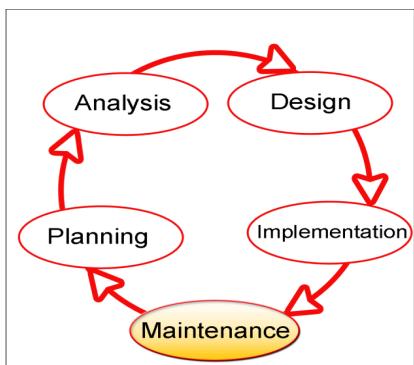
Software engineering is the process of analyzing user needs and designing, constructing, and testing end user applications that will satisfy the needs through the use of software programming languages.

SDLC is the software development process used for development of software, from inception to product release.

The purpose of SDLC is to produce fully-functional, effective, easy-to-use, and easily maintainable software within the estimated time and cost.

SDLC is considered the foundation for all software development methodologies like Waterfall, Iterative, Agile, and so on.

It includes seven phases:



1. Planning
2. Analysis
3. Design
4. Development
5. Testing
6. Deployment
7. Maintenance.

To develop software for such devices, you need to follow the SDLC and begin with proper planning.

Then, gather and analyze requirements like the need for a simple and intuitive interface, remote monitoring, etc.

Next, design the software to address all requirements, focus on the process of making interfaces for user friendliness and develop it.

Test the ready software to see how it works in a refrigerator and address issues.

Finally, deploy the approved software in the next batch of refrigerators and prepare for maintenance. Planning is the first phase.

It mainly focuses on the scope of the problem and device solutions.

The output of this phase includes project plans, schedules, cost estimations, and procurement requirements.

**Software Development Methodology** - Software development methodology is a framework used to divide software development tasks into distinct phases for improving design, product management, and project management. It is also known as Software Development Life Cycle or the SDLC.

## **SDLC MODELS**

1. Waterfall Model
2. Prototype Model
3. Spiral Model
4. Iterative Increment Model
5. V Model

**1.Water fall model:-** The next stage cannot begin without the completion of the initial phase. All steps are requirement ,design ,implementation,verification and maintenance. Takes lot of time.

**2.Iterative model:-** In every single iteration, additional features are designed and added to the software. Each iteration involves coding and testing the software product. There are different phases in an iterative model like inception phase, elaboration phase, construction phase and transition phase. The Inception and Elaboration phases define the project's scope, risks, and architecture, while the Construction and Transition phases focus on coding, testing, and deploying the product to production. Disadvantage:-complete requirement knowledge needed.

**3.Prototype model:-** Before actual software a prototype of the software is created to let users interact with it and from that interaction whatever data we got we use that data to implement in our actual software.

**4.Spiral model:-** It is a combination of both iterative and waterfall models. Does more risk analysis. Four phases involved are objective ,risk analysis, engineering and evaluation.Risk analysis is done using prototype models.

**5.V model:-** Development and testing is done parallelly. Unit testing for Low level design, Integration testing on integrated modules, system testing for the whole system in system design phase , acceptance testing for requirement analysis phase. Not good for ongoing projects.

## Agile

- The agile methodology uses an iterative and team-based approach.
- Its main objective is to quickly deliver the application with complete and functional components.
- In software development, the term agile means the ability to respond to changes.
- It is an iterative and incremental process.

ADLLC Life Cycle:



1. Concept
2. Inception
3. Design, Development, Construction, Testing, and Integration
4. Implementation Deployment
5. Retirement

## Scrum

- Scrum is a subset of Agile. It is a lightweight process framework for agile development, and the most widely-used one.
- Scrum is most often used to manage complex software and product development, using iterative and incremental practices.
- Scrum processes enable organizations to adjust smoothly to rapidly-changing requirements and produce a product that meets evolving business goals.

Phases in Scrum -

1. Initiate
2. Plan and Estimate
3. Implement
4. Review and Retrospect
5. Release

---

Scrum roles and responsibilities:

Team:- Group of individuals who create the project and have diverse skills.

Scrum master:- Person who leads Team.

Product Owner:- Represents stakeholder and customers and they strive for more return of investment through projects.

Note: Burn up chart contains team's work progress in visual form.

Events in scrum:

Sprint:- Basic unit of work for a scrum team.

Sprint planning:- How and what is going to be done in the sprint . Holds before each sprint.

Daily scrum:- To evaluate the progress and trend until the end of sprint . Each day a meeting happens to make sure things are synchronized with plans.

Sprint review:- To know what work has been done till now for future deliveries.

Sprint retrospect:- To learn from the past mistakes, the team reviews the completed sprints.

## Day 2 - Git and Github

[Repository Github Link - <https://github.com/anjalic1902/mthree.git>]

Version Control System(VCS):

Version control system records the changes in a file so that if we don't want a particular change in our code or file we can trace back to previous versions of that file and it helps more people to collaborate easily with each other.

Git is a version control system .

Github is a company which makes tools to interact with git.

What is Git?

- Git is the most used version control system.
- Git tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to.
- Git also makes collaboration easier, allowing changes by multiple people to all be merged into one source.
- Most Git actions only add data to the database, and Git makes it easy to undo changes during the three main states.
- Git has three file states: modified, staged, and committed.

Example :

If we're working on code or a document with a team, version control helps ensure that everyone can work on different features at the same time, track changes, and avoid losing work or overwriting someone else's work.

Features of version control software :

- i. Repository
- ii. Pull Request
- iii. Commit
- iv. Branch
- v. Merge
- vi. Conflict
- vii. Rebase
- viii. Checkout

GITHUB :

GitHub is a cloud-based platform where you can store, share, and work together with others to write code.

Basic Commands :

**1. Initializing a repository:**

a.git init

Initializes a new Git repository in the current directory.

**2. Adding files:**

b. git add <file>

Stages a specific file to be committed.

- git add .

Stages all modified files in the current directory for the next commit.

**3. Committing changes:**

- git commit -m "commit message"

Records the changes with a descriptive message.

- git commit -am "commit message"

Stages and commits all modified files in one command (tracked files only).

**4. Checking the status:**

- git status

Shows the current status of your working directory and the staged area (untracked, modified, staged files).

**5. Viewing commit history:**

- git log

Displays the commit history.

- git log --oneline

Shows a simplified version of the commit history (one commit per line).

**6. Pushing changes to a remote:**

- git push origin <branch>

Pushes the commits to the remote repository's specified branch.

**7. Pulling changes from a remote:**

- 
- `git pull origin <branch>`  
Fetches and merges changes from the remote repository's branch to your local branch.

## 8. Cloning a repository:

- `git clone <repo-url>`  
Downloads an existing Git repository from a remote URL.

## 9. Creating and switching branches:

- `git branch <branch-name>`  
Creates a new branch.
- `git checkout <branch-name>`  
Switches to the specified branch.
- `git checkout -b <branch-name>`  
Creates and switches to a new branch in one command.

## 10. Merging branches:

- `git merge <branch-name>`  
Merges the specified branch into the current branch.

## 11. Resetting changes:

- `git reset --soft HEAD~1`  
Undoes the last commit but keeps the changes staged.
- `git reset --mixed HEAD~1`  
Undoes the last commit and unstages the changes.
- `git reset --hard HEAD~1`  
Discards the last commit and all changes.

## 12. Stashing changes:

- `git stash`  
Temporarily saves your uncommitted changes.
- `git stash pop`  
Restores the stashed changes and removes them from the stash list.

## 13. Deleting branches:

- `git branch -d <branch-name>`  
Deletes the specified branch locally.

- 
- `git push origin --delete <branch-name>`  
Deletes the specified branch from the remote repository.

**Git Rebase** moves your branch's commits to the latest state of another branch, creating a linear history without merge commits.

Key Points:

- It rewrites commit history for a clean timeline.
- Use it to keep your project history tidy and linear.
- Avoid rebasing branches shared with others.

Commands:

Switch to the branch you want to rebase:

bash

1. Rebase your branch onto the target branch (e.g., main):

bash

`git checkout feature`

2. If conflicts occur, resolve them, then continue rebasing:

bash

`git rebase main`

3. Rebase is great for keeping a clean commit history, but use with caution, especially for public branches.

`git add .      git rebase --continue`

## **Git Command used:**

```
106 ls -lrt  
107 cd sdlc/  
108 vi a.txt  
109 git status  
110 vi b.txt  
111 vi c.txt  
112 git status  
113 ls -lrt  
114 git branch -d test1
```

```
115 cd apl
116 git status
117 git push
118 git add --all
119 git status
120 ssh -T git@github.com
121 cd ..
122 git status
123 git checkout main
124 cd sdlc/test1
125 cd sdlc
126 ls -lrt
127 mkdir apl
128 cd apl
129 vi d.txt
130 git status
131 git add --all
132 git status
133 git commit -m "Start a feature"
134 git checkout main
135 vi aplg.txt
136 git merge test1
137 git push
138 ls -lrt
139 git branch -d test1
140 ls -lrt
141 cd apl
142 ls -lrt
143 vi d.txt
144 cd ..
145 git add
146 git add.
147 git add .
148 git branch -d "test merge conflict"
149 git checkout test1
150 git branch -d "test merge conflict"
151 git branch -d "test-merge-conflict"
152 ls -lrt
```

```
153 ls -lrt
154 git branch -b "test-merge-conflict"
155 git branch -d "test-merge-conflict"
156 cd apl
157 ls -lrt
158 git commit -m "changes to be committed"
159 git checkout main
160 git commit
161 git checkout main
162 apl/.d.txt.swp
163 git checkout main
164 vi d.txt
165 git add --all
166 git commit
167 git commit -m "changes to be committed"
168 git checkout main
169 git add --all
170 git commit -m "changes to be committed"
171 git merge test-merge-conflicts
172 git merge test-merge-conflict
173 vi d.txt
174 git commit -m "merging changes"
175 git add -all
176 git add --all
177 git push
178 git checkout -b "test-merge-conflict"
179 vi d.txt
180 git checkout -b "test-merge-conflict"
181 git checkout "test-merge-conflict"
182 git add .
183 git commit -m "merging changes"
184 git checkout "test-merge-conflict"
185 git commit -m "changes"
186 git add
187 git checkout "test-merge-conflict"
188 ls -lrt
189 vi d.text
190 git add .
```

---

```
191 git commit -m "changes"
192 git checkout main
193 git merge test-merge-conflict
194 git log --merge
195 git diff
196 git add
197 git add .
198 git commit -m "changes"
199 ls -lrt
200 git checkout test1
201 ls -lrt
202 git checkout apl
203 ls -lrt
204 git checkout aplg
205 ls -lrt
206 vi a.txt
207 git status
208 git pull
209 git fetch origin main
210 git status
211 git diff main origin/main
212 git rebase origin/main
213 git add .
214 git commit -m "changes"
215 git rebase origin/main
216 git push
217 history
218 mkdir essay_project
219 cd essay_project/
220 git log --graph --oneline -all
221 git log --graph --all
222 git log --graph --oneline -all
223 git log --graph --oneline --all
224 echo "1. Introduction hook engaging open statement background thesis 2. body point 1 supporting arg - evidence - analysis 3. conclusion"> essay.txt
225 vi essay.txt
226 git commit -am "Writing in detail"
227 git log --graph --oneline --all
```

---

```
228 git reset --soft HEAD~1
229 git log --graph --oneline --all
230 echo "title my amazing essay "1. Introduction hook engaging open statement background
thesis 2. body point 1 supporting arg - evidence - analysis 3. conclusion"> essay.txt
231 git commit -am "final boarding"
232 git log --graph --oneline --all
233 git push
234 git pull
235 git push
236 mkdir gitTutorial
237 cd gitTutorial/
238 echo "initial context">file.txt
239 git add file.txt
240 git commit -m "First file add"
241 git checkout -b feature
242 echo "geature work"> file.txt
243 ls -lrt
244 git status
245 git commit -am "feature progress"
246 echo "unfinished feature work" >> file.txt
247 git stash save "unfinished feature"
248 vi file.txt
249 git status
250 git log --oneline
251 git checkout main
252 vi file.txt
253 echo "main branch change">>file.txt
254 vi file.txt
255 git commit -am "changes to file"
256 git checkout feature
257 vi file.txt
258 git stash pop
259 vi file.txt
260 git commit -am "complete feature"
261 git rebase main
262 vi file.txt
263 git status
264 git log --oneline
```

```
265 git commit -am "change final"
266 git push
267 git rebase --continue
268 git push
269 git rebase origin/main
270 git push
271 git push --set-upstream origin feature
272 git commit -help
```

## Activity: Set up a personal Github Repository

```
Anjali@Anjali MINGW64 ~/git (new-feature)
$ git clone https://github.com/anjalic1902/Project.git
Cloning into 'Project'...
Unhandled Exception: System.IO.FileNotFoundException: Could not load file or assembly 'Atlassian.Bitbucket, Version=2.5.1.0, Culture=neutral, PublicKeyToken=null' or one of its dependencies. The system cannot find the file specified.
   at GitCredentialManager.Program.AppMain(Object o)
   at System.Threading.ThreadHelper.ThreadStart_Context(Object state)
   at System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, Object state, Boolean preserveSyncCtx)
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state, Boolean preserveSyncCtx)
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state)
   at System.Threading.ThreadHelper.ThreadStart(Object obj)
anjalic1902
Unhandled Exception: System.IO.FileNotFoundException: Could not load file or assembly 'Atlassian.Bitbucket, Version=2.5.1.0, Culture=neutral, PublicKeyToken=null' or one of its dependencies. The system cannot find the file specified.
   at GitCredentialManager.Program.AppMain(Object o)
   at System.Threading.ThreadHelper.ThreadStart_Context(Object state)
   at System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, Object state, Boolean preserveSyncCtx)
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state, Boolean preserveSyncCtx)
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state)
   at System.Threading.ThreadHelper.ThreadStart(Object obj)
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.

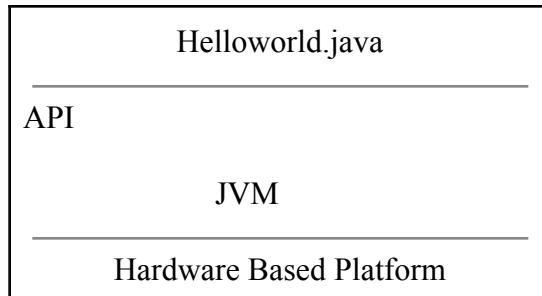
Anjali@Anjali MINGW64 ~/git (new-feature)
$ cd Project

Anjali@Anjali MINGW64 ~/git/Project (main)
$ ls
'Mthree Training Weeks' README.md

Anjali@Anjali MINGW64 ~/git/Project (main)
$ |
```

## Day 3 - Java Basics and Bank Application Group Activity

HelloWorld.java — compiler — HelloWorld.class — Java VM — Computer



- Cursor - Install Extensions - Debugger for Java and Extensions pack for java

### Program 1- HelloWorld

The screenshot shows a Java code editor with the file 'P1HelloWorld.java' open. The code contains a single class definition with a main method that prints "Hello World". Below the code editor is a terminal window showing the execution of the program. The terminal output shows the command 'java P1HelloWorld' being run, followed by the output 'Hello World'.

```
J P1HelloWorld.java > P1HelloWorld > main(String[])
1 public class P1HelloWorld {
2
3     Run | Debug
4     public static void main(String[] args) {
5
6         System.out.println("Hello World");
7     }
8 }
```

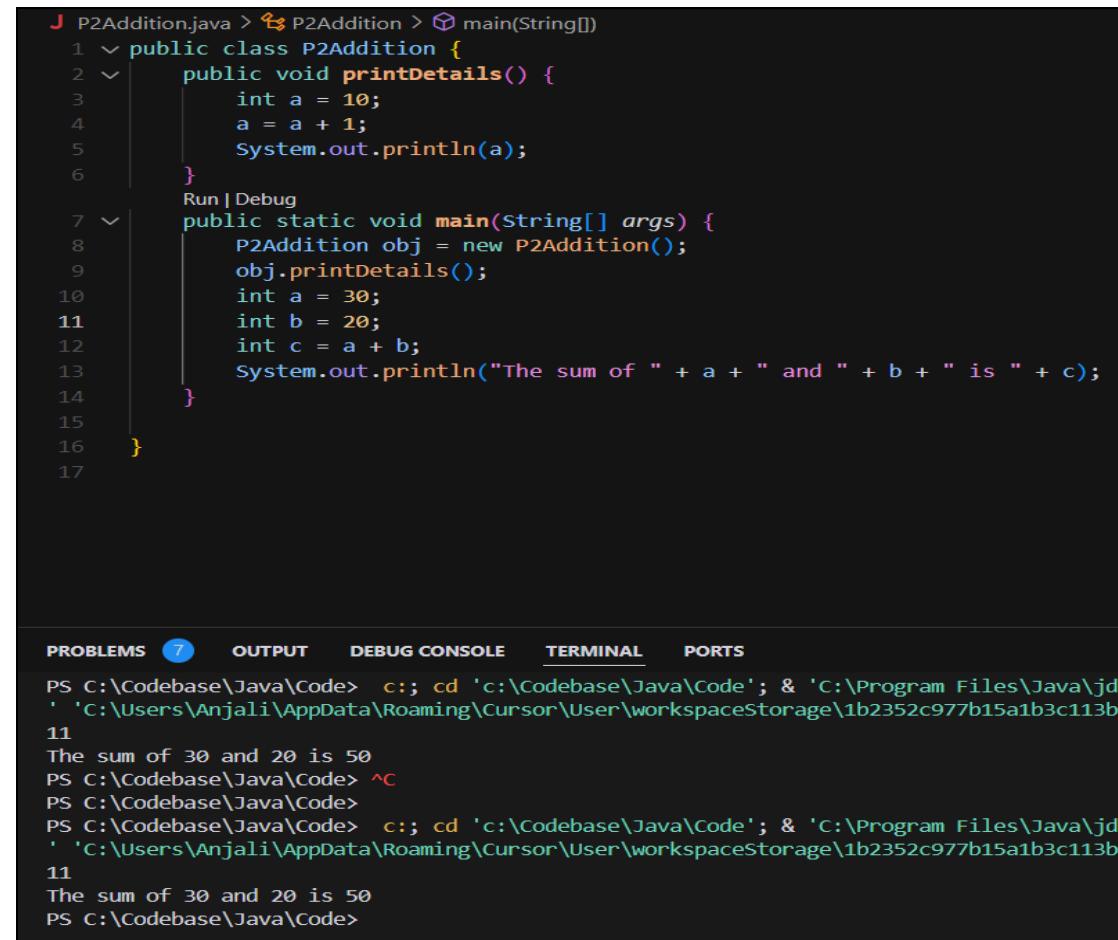
```
PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Codebase\Java\Code> ^C
PS C:\Codebase\Java\Code>
PS C:\Codebase\Java\Code> c;; cd 'c:\Codebase\Java\Code';
'C:\Users\Anjali\AppData\Roaming\Cursor\User\workspaceSto
Hello World
PS C:\Codebase\Java\Code> ^C
PS C:\Codebase\Java\Code>
PS C:\Codebase\Java\Code> c;; cd 'c:\Codebase\Java\Code';
'C:\Users\Anjali\AppData\Roaming\Cursor\User\workspaceSto
Hello World
PS C:\Codebase\Java\Code>
```

**JVM** - The JVM (Java Virtual Machine) is a platform-independent runtime environment that executes Java bytecode. It converts bytecode into machine-specific instructions, enabling Java programs to run on any system with a JVM, regardless of the underlying operating system.

**JRE** - The JRE (Java Runtime Environment) includes the JVM along with the core libraries and other components required to run Java applications. It provides the runtime environment but does not include development tools like a compiler, making it suitable for users who only need to execute Java programs.

**JDK** - The JDK (Java Development Kit) is a complete toolkit for Java developers. It includes the JRE, along with additional tools like the Java compiler (`javac`), debugger, and other utilities for writing, compiling, and debugging Java code. The JDK is necessary for developing Java applications, while the JRE is for running them.

## Program 2 - Addition of Two Numbers



The screenshot shows a Java code editor with the file `P2Addition.java` open. The code defines a class `P2Addition` with a `main` method and a `printDetails` method. The `main` method creates an object of `P2Addition`, calls `printDetails`, and then calculates and prints the sum of `a` and `b`. Below the code editor is a terminal window showing the command to run the program and its output.

```
1 J P2Addition.java > P2Addition > main(String[])
  1 public class P2Addition {
  2     public void printDetails() {
  3         int a = 10;
  4         a = a + 1;
  5         System.out.println(a);
  6     }
  7     public static void main(String[] args) {
  8         P2Addition obj = new P2Addition();
  9         obj.printDetails();
 10        int a = 30;
 11        int b = 20;
 12        int c = a + b;
 13        System.out.println("The sum of " + a + " and " + b + " is " + c);
 14    }
 15
 16 }
 17

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Codebase\Java\Code> cd 'c:\Codebase\Java\Code'; & 'C:\Program Files\Java\jd
' 'C:\Users\Anjali\AppData\Roaming\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b
11
The sum of 30 and 20 is 50
PS C:\Codebase\Java\Code> ^C
PS C:\Codebase\Java\Code>
PS C:\Codebase\Java\Code> cd 'c:\Codebase\Java\Code'; & 'C:\Program Files\Java\jd
' 'C:\Users\Anjali\AppData\Roaming\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b
11
The sum of 30 and 20 is 50
PS C:\Codebase\Java\Code>
```

- Instance - global variables
- Methods - local variables
- Main - static variables

## Methods

```
<access modifier> static <return type> <method name> (<parameter list>) <exception list> {
    <method body>
}
```

## Program 3 - Displaying the data of an Employee

```
J P3Employee.java > P3Employee > P3Employee()
1
2
3 public class P3Employee {
4     public String name;
5     public int age;
6     public String city;
7     public double salary;
8     public String companyName;
9     public String employeeId;
10    public String department;
11    public String designation;
12
13    public void printDetails() {
14        System.out.println("Name: " + name);
15        System.out.println("Age: " + age);
16        System.out.println("City: " + city);
17        System.out.println("Salary: " + salary);
18        System.out.println("Company Name: " + companyName);
19        System.out.println("Employee ID: " + employeeId);
20        System.out.println("Department: " + department);
21        System.out.println("Designation: " + designation);
22    }
23
24    public void setDetails(String name, int age, String city, double salary, String companyName, String employeeId, String department, String designation) {
25        this.name = name;
26        this.age = age;
27        this.city = city;
28        this.salary = salary;
29        this.companyName = companyName;
30        this.employeeId = employeeId;
31        this.department = department;
32        this.designation = designation;
33    }
34    public P3Employee() {
35        name = "Anjali Chauhan";
36        age = 22;
37        city = "Rawatbhata";
38        salary = 100000;
39        companyName = "mthree";
40        employeeId = "0810CA221024";
41        department = "IT";
42        designation = "Software Developer Trainee";
43        printDetails();
44    }
45    public P3Employee(String name, int age, String city, double salary, String companyName, String employeeId, String department, String designation) {
46        this.name = name;
47        this.age = age;
48        this.city = city;
49        this.salary = salary;
50        this.companyName = companyName;
51        this.employeeId = employeeId;
52        this.department = department;
53        this.designation = designation;
54        printDetails();
55    }
56
57
58    Run|Debug
59    public static void main(String[] args) {
60        P3Employee obj = new P3Employee();
61        obj.setDetails(name:"Jasa", age:30, city:"New York", salary:100000, companyName:"ABC Company", employeeId:"1234567890", "IT", "Software Engineer");
62        obj.printDetails();
63        P3Employee obj2 = new P3Employee(name:"Jasa", age:30, city:"New York", salary:100000, companyName:"ABC Company", employeeId:"1234567890", "IT", "Software Engineer");
64        obj2.printDetails();
65    }
}
```

```

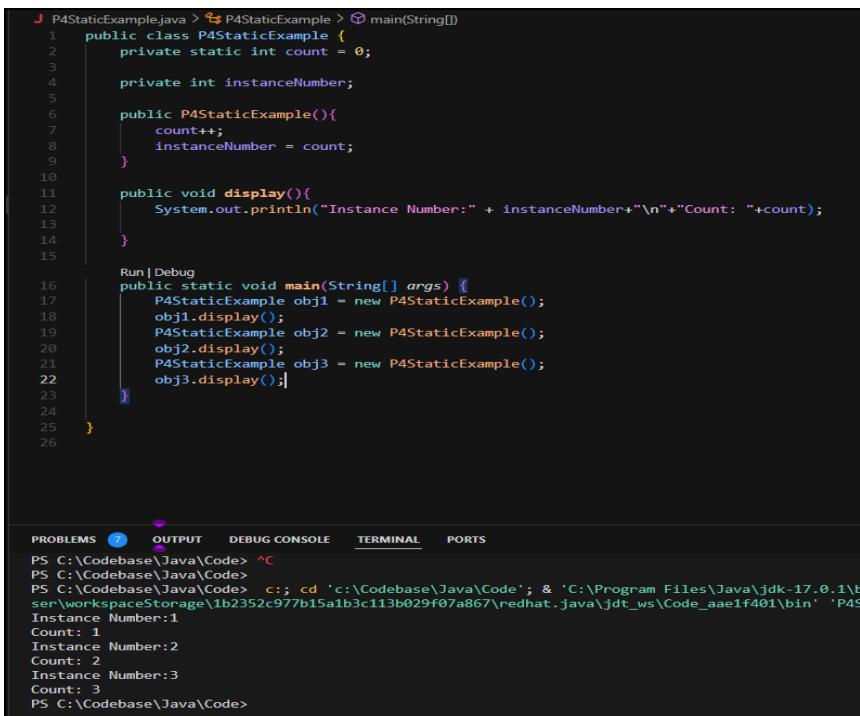
Name: Anjali Chauhan
Age: 22
City: Rawatbhata
Salary: 100000.0
Company Name: mthree
Employee ID: 0810CA221024
Department: IT
Designation: Software Developer Trainee
Name: Jasa
Age: 30
City: New York
Salary: 100000.0
Company Name: ABC Company
Employee ID: 1234567890
Department: IT
Designation: Software Engineer
Name: Jasa
Age: 30
City: New York
Salary: 100000.0
Company Name: ABC Company
Employee ID: 1234567890
Department: IT
Designation: Software Engineer
Name: Jasa
Age: 30
City: New York
Salary: 100000.0
Company Name: ABC Company
Employee ID: 1234567890
Department: IT
Designation: Software Engineer
PS C:\Codebase\Java\Code>

```

## Comments

- Single-Line
- Multi-Line
- Doc

## Program 4 - Demonstration of Static



```

J P4StaticExample.java > P4StaticExample > main(String[])
1  public class P4StaticExample {
2      private static int count = 0;
3
4      private int instanceNumber;
5
6      public P4StaticExample(){
7          count++;
8          instanceNumber = count;
9      }
10
11     public void display(){
12         System.out.println("Instance Number: " + instanceNumber+"\n"+ "Count: " +count);
13     }
14
15     Run | Debug
16     public static void main(String[] args) {
17         P4StaticExample obj1 = new P4StaticExample();
18         obj1.display();
19         P4StaticExample obj2 = new P4StaticExample();
20         obj2.display();
21         P4StaticExample obj3 = new P4StaticExample();
22         obj3.display();
23     }
24
25 }
26

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Codebase\Java\Code> ^C
PS C:\Codebase\Java\Code> PS C:\Codebase\Java\Code> c::: cd 'c:\Codebase\Java\Code'; & 'C:\Program Files\Java\jdk-17.0.1\bin' 'P4S
ser\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P4S
Instance Number:1
Count: 1
Instance Number:2
Count: 2
Instance Number:3
Count: 3
PS C:\Codebase\Java\Code>

```

---

## **Data Types**

Primitive Data Types:

- byte: 8-bit signed integer (-128 to 127)
- short: 16-bit signed integer (-32,768 to 32,767)
- int: 32-bit signed integer (-2,147,483,648 to 2,147,483,647)
- long: 64-bit signed integer (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
- float: 32-bit floating-point number
- double: 64-bit floating-point number
- char: 16-bit Unicode character
- boolean: Represents true or false values

Non-Primitive Data Types:

- String: Represents a sequence of characters
- Arrays: Stores a collection of elements of the same data type
- Classes: User-defined data types that encapsulate data and behavior
- Interfaces: Define a contract for classes to implement
- Enums: Represents a fixed set of constants

### **Program 5 - To demonstrate the Type Casting concepts**

```

J P5WideningTypeCasting.java > PSWideningTypeCasting
1 public class P5WideningTypeCasting {
2     Run | Debug
3     public static void main(String[] args) {
4         // Starting with byte
5         byte byteValue = 10;
6         System.out.println("Original byte value: " + byteValue);
7
8         // byte to short
9         short shortValue = byteValue;
10        System.out.println("byte to short: " + shortValue);
11
12        // short to char
13        // Note: This is not a direct widening conversion, as char is unsigned
14        // We'll use a positive short value to demonstrate
15        short positiveShort = 65; // ASCII value for 'A'
16        char charValue = (char) positiveShort;
17        System.out.println("short to char: " + charValue);
18
19        // char to int
20        int intFromChar = charValue;
21        System.out.println("char to int: " + intFromChar);
22
23        // int to long
24        long longValue = intFromChar;
25        System.out.println("int to long: " + longValue);
26
27        // Long to float
28        float floatValue = longValue;
29        System.out.println("long to float: " + floatValue);
30
31        // float to double
32        double doubleValue = floatValue;
33        System.out.println("float to double: " + doubleValue);
34
35        // Demonstrating multiple steps of widening in one assignment
36        int bigInt = 1234567;
37        double doubleFromInt = bigInt;
38        System.out.println("int directly to double: " + doubleFromInt);
39
40        // Demonstrating widening in expressions
41        byte small = 10;
42        short medium = 100;
43        int result = small + medium; // byte and short are promoted to int
44        System.out.println("Result of byte + short (as int): " + result);
45
46        // Widening with Literals
47        int largeInt = 2_000_000_000; // 2 billion
48        long veryLarge = largeInt * 3L; // Result is too large for int, so it's widened to Long
49        System.out.println("Large calculation result (as long): " + veryLarge);
50
51        // Demonstrating potential loss of precision
52        long veryPrecise = 123456789123456789L;
53        float lessPrecise = veryPrecise; // Potential loss of precision
54        System.out.println("Long to float (potential precision loss): " + lessPrecise);
}

```

```

Original byte value: 10
byte to short: 10
short to char: A
char to int: 65
int to long: 65
long to float: 65.0
float to double: 65.0
int directly to double: 1234567.0
Result of byte + short (as int): 110
Large calculation result (as long): 6000000000
Long to float (potential precision loss): 1.23456791E17
PS C:\Codebase\Java\Code>

```

## Program 6 - Working with Operators

```
J P6ComprehensiveLogicalOperators.java > P6ComprehensiveLogicalOperators > main(String[])
1  public class P6ComprehensiveLogicalOperators {
2      Run|Debug
3      public static void main(String[] args) {
4          // Basic boolean variables
5          boolean t = true;
6          boolean f = false;
7
7          System.out.println("1. Basic Logical Operations:");
8          System.out.println("    AND: true && true = " + (t && t));
9          System.out.println("    AND: true && false = " + (t && f));
10         System.out.println("    AND: false && true = " + (f && t));
11         System.out.println("    AND: false && false = " + (f && f));
12         System.out.println("    OR: true || true = " + (t || t));
13         System.out.println("    OR: true || false = " + (t || f));
14         System.out.println("    OR: false || true = " + (f || t));
15         System.out.println("    OR: false || false = " + (f || f));
16         System.out.println("    NOT: !true = " + (!t));
17         System.out.println("    NOT: !false = " + (!f));
18
19         System.out.println("\n2. Short-circuit Evaluation:");
20         System.out.println("    false && (1/0 > 0) = " + (f && (1/0 > 0))); // No ArithmeticException
21         System.out.println("    true || (1/0 > 0) = " + (t || (1/0 > 0))); // No ArithmeticException
22
23         System.out.println("\n3. Operator Precedence:");
24         System.out.println("    true || false && false = " + (t || f && f)); // && has higher precedence
25         System.out.println("    (true || false) && false = " + ((t || f) && f)); // Parentheses change precedence
26
27         System.out.println("\n4. Combining with Comparison Operators:");
28         int x = 5, y = 10;
29         System.out.println("    (x < y) && (y > 0) = " + ((x < y) && (y > 0)));
30         System.out.println("    (x > y) || (y < 20) = " + ((x > y) || (y < 20)));
31
32         System.out.println("\n5. Complex Conditions:");
33         boolean a = true, b = false, c = true;
34         System.out.println("    (a && b) || (a && c) = " + ((a && b) || (a && c)));
35         System.out.println("    a && (b || c) = " + (a && (b || c)));
36         System.out.println("    !a || (b && !c) = " + (!a || (b && !c)));
37
38         int a1=10, b1=10;
39         b1 = (a1 == 10)? 20:30;
40         System.out.println("b1 = " + b1);
41
42         System.out.println("\n6. Bitwise vs. Logical Operators:");
43         System.out.println("    true & false = " + (t & f)); // Bitwise AND
44         System.out.println("    true | false = " + (t | f)); // Bitwise OR
45         System.out.println("    true ^ false = " + (t ^ f)); // Bitwise XOR
46
47         System.out.println("\n7. Short-circuit vs. Non-short-circuit:");
48         int i = 0;
49         boolean result1 = (f && (++i > 0)); // i is not incremented
50         boolean result2 = (f & (++i > 0)); // i is incremented
51         System.out.println("    Short-circuit AND result: " + result1 + ", i = " + i);
52         System.out.println("    Non-short-circuit AND result: " + result2 + ", i = " + i);
53
54         System.out.println("\n8. Logical Operators with Non-boolean Operands:");


```

```

System.out.println("  (1 < 2) && (3 < 4) = " + ((1 < 2) && (3 < 4)));
System.out.println("  ('a' < 'b') || ('c' > 'd') = " + (('a' < 'b') || ('c' > 'd')));

System.out.println(x:"\n9. Logical Operators in Control Structures:");
if (t && !f) {
    System.out.println(x:"  This will be printed.");
}

int j = 0;
while (j < 3 && t) {
    System.out.println("  j = " + j);
    j++;
}

System.out.println(x:"\n10. Logical Operators with Method Calls:");
System.out.println("  isPositive(5) && isEven(4) = " + (isPositive(5) && isEven(4)));
System.out.println("  isPositive(-3) || isEven(7) = " + (isPositive(-3) || isEven(7)));

System.out.println(x:"\n11. Logical Operators with Null Checks:");
String str = null;
System.out.println("  (str != null) && (str.length() > 0) = " + ((str != null) && (str.length() > 0)));
// System.out.println("  (str.length() > 0) && (str != null) = " + ((str.length() > 0) && (str != null)))

System.out.println(x:"\n12. Using Logical Operators for Conditional Assignment:");
int value = t ? 1 : 0;
System.out.println("  value = " + value);

System.out.println(x:"\n13. Logical Operators in Lambda Expressions:");
java.util.function.Predicate<Integer> isPositiveAndEven = n -> n > 0 && n % 2 == 0;
System.out.println("  Is 6 positive and even? " + isPositiveAndEven.test(t:6));
System.out.println("  Is 5 positive and even? " + isPositiveAndEven.test(t:5));
}

private static boolean isPositive(int n) {
    return n > 0;
}
Ctrl+L to chat, Ctrl+K to generate
private static boolean isEven(int n) {
    return n % 2 == 0;
}

```

```
1. Basic Logical Operations:  
    AND: true && true = true  
    AND: true && false = false  
    AND: false && true = false  
    AND: false && false = false  
    OR: true || true = true  
    OR: true || false = true  
    OR: false || true = true  
    OR: false || false = false  
    NOT: !true = false  
    NOT: !false = true  
  
2. Short-circuit Evaluation:  
    false && (1/0 > 0) = false  
    true || (1/0 > 0) = true  
  
3. Operator Precedence:  
    true || false && false = true  
    (true || false) && false = false  
  
4. Combining with Comparison Operators:  
    (x < y) && (y > 0) = true  
    (x > y) || (y < 20) = true  
  
5. Complex Conditions:  
    (a && b) || (a && c) = true  
    a && (b || c) = true  
    !a || (b && !c) = false  
b1 = 20  
  
6. Bitwise vs. Logical Operators:  
    true & false = false  
    true | false = true  
    true ^ false = true  
  
7. Short-circuit vs. Non-short-circuit:  
    Short-circuit AND result: false, i = 1  
    Non-short-circuit AND result: false, i = 1  
  
8. Logical Operators with Non-boolean Operands:  
    (1 < 2) && (3 < 4) = true  
    ('a' < 'b') || ('c' > 'd') = true  
  
9. Logical Operators in Control Structures:  
    This will be printed.  
    j = 0  
    j = 1  
    j = 2  
  
10. Logical Operators with Method Calls:  
    isPositive(5) && isEven(4) = true  
    isPositive(-3) || isEven(7) = false  
  
11. Logical Operators with Null Checks:  
    (str != null) && (str.length() > 0) = false  
  
12. Using Logical Operators for Conditional Assignment:  
    value = 1  
  
13. Logical Operators in Lambda Expressions:  
    Is 6 positive and even? true  
    Is 5 positive and even? false  
PS C:\Codebase\Java\Code>
```

**Program 7 - InstanceOf Operator:** The instanceof operator in Java is used to test whether an object is an instance of a specific class or implements a specific interface. It returns true if the object is an instance of the class or interface, and false otherwise.

```
J P7Car.java > ...
1  class Vehicle {}
2  //These class explains the use of instanceof operator
3
4  public class P7Car extends Vehicle{
5      Run | Debug
6      public static void main(String[] args) {
7          Vehicle a = new P7Car();
8          boolean result = a instanceof P7Car;
9          System.out.println("result = " + result);
10     }
11 }
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> ^C
PS C:\Codebase\Java\Code>
PS C:\Codebase\Java\Code> c:; cd 'c:\Codebase\Java\Code'; & 'C:\P
' 'C:\Users\Anjali\AppData\Roaming\Cursor\User\workspaceStorage\1b
result = true
PS C:\Codebase\Java\Code>
```

## Switch Case Statement

- Returns a constant value
- The constant could be either byte, short, int or string
- Can be used only for equality checking

```
switch(variable/expression)
{
    case value1:
        //statements
        break;
    case value2:
        //statements
        break; . . . . .
    default:
        //statements
}
```

## Program 8 - Using of Switch Case and user input using scanner class

```
J P8Input.java > 🏃 P8Input
1
2 import java.util.Scanner;
3
4
5 public class P8Input {
6     Run|Debug
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9         System.out.println("Enter your name: ");
10        String name = sc.nextLine();
11        System.out.println("Hello, " + name + "!");
12
13        //Use Scanner to get a calculator which take in
14        System.out.println("Enter first number: ");
15        double num1 = sc.nextDouble();
16        System.out.println("Enter second number: ");
17        double num2 = sc.nextDouble();
18        System.out.println("Enter operation: ");
19        char operation = sc.next().charAt(index:0);
20        double result = 0;
21        switch (operation) {
22            case '+':
23                result = num1 + num2;
24                break;
25            case '-':
26                result = num1 - num2;
27                break;
28            case '*':
29                result = num1 * num2;
30                break;
31            }
32        System.out.println("Result: " + result);
33    }
34 }
```

## Program 9 - Group Activity - Banking Application

```
J P9BankApplication.java > P9BankApplication > main(String[])
1 import java.util.Scanner;
2
3 class BankAccount {
4     private String accHolderName;
5     private String accNumber;
6     private double balance;
7
8     // To create a new account
9     public BankAccount(String accHolderName, String accNumber, double balance) {
10         this.accHolderName = accHolderName;
11         this.accNumber = accNumber;
12         this.balance = balance;
13     }
14
15     // For deposit money
16     public void deposit(double amount) {
17         if (amount > 0) {
18             balance += amount;
19             System.out.println("Deposited: " + amount);
20         } else {
21             System.out.println("Invalid");
22         }
23     }
24
25     // For withdraw money
26     public void withdraw(double amount) {
27         if (amount > 0 && amount <= balance) {
28             balance -= amount;
29             System.out.println("Withdraw: " + amount);
30         } else {
31             System.out.println("Invalid");
32         }
33     }
34
35     // To display balance
36     public void checkBalance() {
37         System.out.println("Balance: " + balance);
38     }
39
40     // To display account information
41     public void accInfo() {
42         System.out.println("Account Holder: " + accHolderName);
43         System.out.println("Account Number: " + accNumber);
44         System.out.println("Balance: " + balance);
45     }
46 }
47
48 public class P9BankApplication {
49     Run | Debug
50     public static void main(String[] args) {
51         Scanner sc = new Scanner(System.in);
52
53         // Create a new bank account
54         System.out.println("Enter Account Holder Name: ");
55         String name = sc.nextLine();
56
57         System.out.println("Enter Account Number: ");
58         String accNumber = sc.nextLine();
59
60         System.out.println("Enter Initial Balance: ");
61         double balance = sc.nextDouble();
62
63         BankAccount account = new BankAccount(name, accNumber, balance);
64     }
65 }
```

```
62     BankAccount account = new BankAccount(name, accNumber, balance);
63
64
65     int option;
66     do {
67         System.out.println("\nBank Application:");
68         System.out.println("1. Deposit");
69         System.out.println("2. Withdraw");
70         System.out.println("3. Check Balance");
71         System.out.println("4. Account Information");
72         System.out.println("5. Exit");
73         System.out.print("Choose an option: ");
74         option = sc.nextInt();
75
76         switch (option) {
77             case 1:
78                 System.out.print("Enter amount to deposit: ");
79                 double depositAmount = sc.nextDouble();
80                 account.deposit(depositAmount);
81                 break;
82             case 2:
83                 System.out.print("Enter amount to withdraw: ");
84                 double withdrawAmount = sc.nextDouble();
85                 account.withdraw(withdrawAmount);
86                 break;
87             case 3:
88                 account.checkBalance();
89                 break;
90             case 4:
91                 account.accInfo();
92                 break;
93             case 5:
94                 System.out.println("Exit");
95                 break;
96             default:
97                 System.out.println("Invalid");
98         }
99     } while (option != 5);
100
101     sc.close();
102 }
103
104 }
```

## Day 4 - Java Continue and Activity

Working with Control Flow Statements:

1. Conditional Statements
  - 1.1. if Statement
  - 1.2. if-else Statement
  - 1.3. if-else-if Statement
  - 1.4. Nested-if Statement
  - 1.5. Switch Statement
2. Looping Statements
  - 2.1. for Loop
  - 2.2. Enhanced for Loop
  - 2.3. While Loop
  - 2.4. do-while Loop
3. Jump Statements
  - 3.1. break Statement
  - 3.2. continue Statement
  - 3.3. return Statement
4. Exception Statements
  - 4.1. try-catch
  - 4.2. try-catch-finally
  - 4.3. try-with-resources
5. Assertions

## Program - 10 - Using of all Control Statements

```
J P10JavaControlStatements.java > ...
1  import java.io.BufferedReader;
2  import java.io.FileReader;
3  import java.io.IOException;
4  import java.util.Arrays;
5  | Ctrl+L to chat, Ctrl+K to generate
6  public class P10JavaControlStatements {
7
8      Run | Debug
9      public static void main(String[] args) {
10         // 1. Conditional Statements
11         conditionalStatements();
12
13         // 2. Looping Statements
14         loopingStatements();
15
16         // 3. Jump Statements
17         jumpStatements();
18
19         // 4. Exception Handling
20         exceptionHandling();
21
22         // 5. Assertions
23         assertions();
24     }
25
26     // 1. Conditional Statements
27     private static void conditionalStatements() {
28         System.out.println("\n--- Conditional Statements ---");
29
30         // 1.1 if statement
31         int age = 18;
32         if (age >= 18) {
33             System.out.println("You are eligible to vote.");
34         }
35
36         // 1.2 if-else statement
37         int score = 75;
38         if (score >= 60) {
39             System.out.println("You passed!");
40         } else {
41             System.out.println("You failed.");
42         }
43
44         // 1.3 if-else-if Ladder
45         int grade = 85;
46         if (grade >= 90) {
47             System.out.println("Grade: A");
```

```
6  public class P10JavaControlStatements {
26      private static void conditionalStatements() {
47          } else if (grade >= 80) {
48              System.out.println("Grade: A");
49          } else if (grade >= 70) {
50              System.out.println("Grade: B");
51          } else if (grade >= 60) {
52              System.out.println("Grade: C");
53          } else {
54              System.out.println("Grade: F");
55          }
56
57      // 1.4 Nested if statements
58      boolean hasLicense = true;
59      if (age >= 18) {
60          if (hasLicense) {
61              System.out.println("You can drive.");
62          } else {
63              System.out.println("You need to get a license first.");
64          }
65      } else {
66          System.out.println("You are too young to drive.");
67      }
68
69      // 1.5 switch statement
70      int dayOfWeek = 3;
71      switch (dayOfWeek) {
72          case 1:
73              System.out.println("Day: Monday");
74              break;
75          case 2:
76              System.out.println("Day: Tuesday");
77              break;
78          case 3:
79              System.out.println("Day: Wednesday");
80              break;
81          case 4:
82              System.out.println("Day: Thursday");
83              break;
84          case 5:
85              System.out.println("Day: Friday");
86              break;
87          case 6:
88          case 7:
89              System.out.println("Day: Weekend");
90              break;
91      default:
```

```
26     private static void conditionalStatements() {
27         |     |     System.out.println("Invalid day");
28     }
29 }
30
31 // 2. Looping Statements
32 private static void loopingStatements() {
33     System.out.println("\n--- Looping Statements ---");
34
35     // 2.1 for Loop
36     System.out.println("For loop:");
37     for (int i = 1; i <= 5; i++) {
38         System.out.println("Iteration: " + i);
39     }
40
41     // 2.2 Enhanced for Loop (for-each)
42     System.out.println("\nEnhanced for loop:");
43     int[] numbers = {1, 2, 3, 4, 5};
44     for (int num : numbers) {
45         System.out.println("Number: " + num);
46     }
47
48     // 2.3 while Loop
49     System.out.println("\nWhile loop:");
50     int count = 0;
51     while (count < 5) {
52         System.out.println("Count: " + count);
53         count++;
54     }
55
56     // 2.4 do-while Loop
57     System.out.println("\nDo-while loop:");
58     int num = 1;
59     do {
60         System.out.println("Number: " + num);
61         num++;
62     } while (num <= 5);
63 }
64
65 // 3. Jump Statements
66 private static void jumpStatements() {
67     System.out.println("\n--- Jump Statements ---");
68
69     // 3.1 break statement
70     System.out.println("Break statement:");
71     for (int i = 1; i <= 10; i++) {
```

```

6   public class P10JavaControlStatements {
131     private static void jumpStatements() {
137       if (i == 6) {
138         break;
139       }
140       System.out.println("Iteration: " + i);
141     }
142
143     // 3.2 continue statement
144     System.out.println(x:"\nContinue statement:");
145     for (int i = 1; i <= 5; i++) {
146       if (i == 3) {
147         continue;
148       }
149       System.out.println("Iteration: " + i);
150     }
151
152     // 3.3 return statement
153     System.out.println(x:"\nReturn statement:");
154     System.out.println("Sum: " + sum(a:5, b:3));
155   }
156
157   private static int sum(int a, int b) {
158     return a + b;
159   }
160
161   // 4. Exception Handling
162   private static void exceptionHandling() {
163     System.out.println(x:"\n--- Exception Handling ---");
164
165     // 4.1 try-catch
166     System.out.println(x:"Try-catch:");
167     try {
168       int result = 10 / 0;
169     } catch (ArithmaticException e) {
170       System.out.println("Error: " + e.getMessage());
171     }
172
173     // 4.2 try-catch-finally
174     System.out.println(x:"\nTry-catch-finally:");
175     try {
176       int[] numbers = {1, 2, 3};
177       System.out.println(numbers[3]);
178     } catch (ArrayIndexOutOfBoundsException e) {
179       System.out.println("Error: " + e.getMessage());
180     } finally {
181       System.out.println(x:"This block always executes\n");

```

```
    System.out.println("This block always executes.");
}

// 4.3 try-with-resources
System.out.println("\nTry-with-resources:");
try (BufferedReader br = new BufferedReader(new FileReader(fileName:"test.txt"))) {
    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    System.out.println("Error reading file: " + e.getMessage());
}
}

// 5. Assertions
private static void assertions() {
    System.out.println("\n--- Assertions ---");
    int age = -5;
    assert age >= 0 : "Age cannot be negative";
    System.out.println("Age: " + age);
}
}
```

```
--- Conditional Statements ---
You are eligible to vote.
You passed!
Grade: B
You can drive.
Day: Wednesday

--- Looping Statements ---
For loop:
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5

Enhanced for loop:
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5

While loop:
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4

Do-while loop:
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5

--- Jump Statements ---
Break statement:
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5

Continue statement:
Iteration: 1
Iteration: 2
Iteration: 4
Iteration: 5

Return statement:
Sum: 8

--- Exception Handling ---
Try-catch:
Error: / by zero

Try-catch-finally:
Error: Index 3 out of bounds for length 3
This block always executes.

Try-with-resources:
Error reading file: test.txt (The system cannot find the file specified)

--- Assertions ---
Age: -5
PS C:\Codebase\Java\Code>
```

## Program 11 - ATM Simulator

```
J P11ATMSimulator.java > P11ATMSimulator
1 import java.util.Scanner;
2 import java.util.Random;
3
4 public class P11ATMSimulator {
5     private static final int PIN = 1234; // Simulated correct PIN
6     private static final int MAX_ATTEMPTS = 3;
7     private static double balance = 1000.00; // Initial balance
8
9     Run | Debug
10    public static void main(String[] args) {
11        Scanner scanner = new Scanner(System.in);
12        Random random = new Random();
13
14        System.out.println("Welcome to the ATM Simulator");
15
16        // PIN verification using do-while loop
17        int attempts = 0;
18        boolean pinVerified = false;
19        do {
20            System.out.print("Please enter your PIN: ");
21            int enteredPin = scanner.nextInt();
22
23            if (enteredPin == PIN) {
24                pinVerified = true;
25                System.out.println("PIN accepted. Access granted.");
26            } else {
27                attempts++;
28                System.out.println("Incorrect PIN. Attempts remaining: " + (MAX_ATTEMPTS - attempts));
29            }
30        } while (!pinVerified && attempts < MAX_ATTEMPTS);
31
32        if (!pinVerified) {
33            System.out.println("Too many incorrect attempts. Your card has been blocked.");
34            return;
35        }
36
37        // Main ATM menu using do-while Loop
38        int choice;
39        do {
40            System.out.println("\nATM Menu:");
41            System.out.println("1. Check Balance");
42            System.out.println("2. Withdraw Money");
43            System.out.println("3. Deposit Money");
```

```
J P11ATMSimulator.java > P11ATMSimulator
 4  public class P11ATMSimulator {
 5      public static void main(String[] args) {
 6          ...
 7          System.out.println("4. Exit");
 8          System.out.print("Enter your choice: ");
 9          choice = scanner.nextInt();
10
11          switch (choice) {
12              case 1:
13                  System.out.printf("Your current balance is: $%.2f\n", balance);
14                  break;
15              case 2:
16                  System.out.print("Enter amount to withdraw: $");
17                  double withdrawAmount = scanner.nextDouble();
18                  if (withdrawAmount > balance) {
19                      System.out.println("Insufficient funds.");
20                  } else {
21                      balance -= withdrawAmount;
22                      System.out.printf("%.2f withdrawn. New balance: $%.2f\n", withdrawAmount, balance);
23                  }
24                  break;
25              case 3:
26                  System.out.print("Enter amount to deposit: $");
27                  double depositAmount = scanner.nextDouble();
28                  balance += depositAmount;
29                  System.out.printf("%.2f deposited. New balance: $%.2f\n", depositAmount, balance);
30                  break;
31              case 4:
32                  System.out.println("Thank you for using the ATM. Goodbye!");
33                  break;
34              default:
35                  System.out.println("Invalid choice. Please try again.");
36          }
37
38          // Simulate processing time
39          try {
40              Thread.sleep(random.nextInt(1000) + 500);
41          } catch (InterruptedException e) {
42              e.printStackTrace();
43          }
44
45      } while (choice != 4);
46
47      scanner.close();
48  }
```

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java  
13b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin'  
Welcome to the ATM Simulator  
Please enter your PIN: 123  
Incorrect PIN. Attempts remaining: 2  
Please enter your PIN: 1234  
PIN accepted. Access granted.  
  
ATM Menu:  
1. Check Balance  
2. Withdraw Money  
3. Deposit Money  
4. Exit  
Enter your choice: 1  
Your current balance is: $1000.00  
  
ATM Menu:  
1. Check Balance  
2. Withdraw Money  
3. Deposit Money  
4. Exit  
Enter your choice: 2  
Enter amount to withdraw: $500  
$500.00 withdrawn. New balance: $500.00  
  
ATM Menu:  
1. Check Balance  
2. Withdraw Money  
3. Deposit Money  
4. Exit  
Enter your choice: 3  
Enter amount to deposit: $1000  
$1000.00 deposited. New balance: $1500.00  
  
ATM Menu:  
1. Check Balance  
2. Withdraw Money  
3. Deposit Money  
4. Exit  
Enter your choice: 4  
Thank you for using the ATM. Goodbye!  
PS C:\Codebase\Java\Code>
```

---

## **Classes, Objects and Creating New Types:**

Types (classes) in Java simply consist of fields (or properties) and behaviors (or **methods**). Fields and behaviors are sometimes referred to as members.

A common technique used to achieve data hiding in Java is the use of accessors and mutators (these are also known as getters and setters in Java). Accessors and mutators are simply **methods** that get and set (respectively) the values of the properties (or fields) on an object.

### **Dot Operator:**

The dot operator ( . ) is used to access public properties or **methods** of an object. The dot operator is used for static and non-static properties and **methods**. On the left side of the dot operator is the class name (for static fields and **methods**) or the variable name of the instance (for non-static fields or **methods**). On the right side of the dot operator is the method or field we want to access.

### **The this keyword:**

The **this** keyword is used to refer to the instance of the class in which the code is currently executing. It is used in conjunction with the dot operator to access properties and **methods** of the containing class.

### **Constructor:**

A constructor is a special method that is called when you create an instance of your class. Constructors are usually used to initialize the properties of a newly-instantiated object. Although constructors are **methods**, there are some special rules that must be followed when creating a constructor:

- A constructor must have the same name as the class that it is a part of. For example, the constructor for a class called Dog would be Dog().
- Constructors never have a return type, not even void.
- Constructors can have parameters but don't have to.
- There can be more than one constructor in a given class.

---

## Program 12: Demonstrating OOPs Concepts:

1. **Classes and Objects:** A class is a blueprint for creating objects. It defines attributes (fields) and behaviors (methods) that the objects of the class will have.

```
6
7  // 1. Classes and Objects
8  // A class is a blueprint for creating objects. It defines attributes (fields)
9  // and behaviors (methods) that the objects of the class will have.
10
11 class Car {
12     // Fields (attributes)
13     String brand;
14     String model;
15     int year;
16
17     // Constructor: special method to initialize objects
18     Car(String brand, String model, int year) {
19         this.brand = brand;
20         this.model = model;
21         this.year = year;
22     }
23
24     // Method (behavior)
25     void displayInfo() {
26         System.out.println("Car: " + year + " " + brand + " " + model);
27     }
28 }
29
```

2. **Encapsulation:** Encapsulation is the bundling of data and the methods that operate on that data within a single unit (class).

```

29
30 // 2. Encapsulation
31 // Encapsulation is the bundling of data and the methods that operate on that data within a single unit (class).
32 // It restricts direct access to some of an object's components, which is a means of preventing accidental
33 // interference and misuse of the methods and data.
34
35 class BankAccount {
36     // Private fields - can only be accessed within this class
37     private String accountNumber;
38     private double balance;
39
40     // Constructor
41     public BankAccount(String accountNumber, double initialBalance) {
42         this.accountNumber = accountNumber;
43         this.balance = initialBalance;
44     }
45
46     // Public methods to access and modify the private fields
47     public String getAccountNumber() {
48         return accountNumber;
49     }
50
51     public double getBalance() {
52         return balance;
53     }
54
55     public void deposit(double amount) {
56         if (amount > 0) {
57             balance += amount;
58         }
59     }
60
61     public void withdraw(double amount) {
62         if (amount > 0 && amount <= balance) {
63             balance -= amount;
64         }
65     }
66 }
```

**3. Inheritance:** Inheritance is a mechanism where a new class is derived from an existing class.

```

67
68 // 3. Inheritance
69 // Inheritance is a mechanism where a new class is derived from an existing class.
70 // The new class (subclass) inherits fields and methods from the existing class (superclass).
71
72 // Superclass
73 class Animal {
74     String name;
75
76     Animal(String name) {
77         this.name = name;
78     }
79
80     void eat() {
81         System.out.println(name + " is eating.");
82     }
83 }
84
85 // Subclass
86 class Dog extends Animal {
87     Dog(String name) {
88         super(name); // Call the superclass constructor
89     }
90
91     void bark() {
92         System.out.println(name + " is barking.");
93     }
94 }
```

- 4. Polymorphism:** Polymorphism allows objects of different classes to be treated as objects of a common superclass.

```
96 // 4. Polymorphism
97 // Polymorphism allows objects of different classes to be treated as objects of a common superclass.
98 // It can be achieved through method overriding and method overloading.
99
100 // Method Overriding: providing a specific implementation of a method in the subclass
101 // that is already defined in the superclass.
102 class Cat extends Animal {
103     Cat(String name) {
104         super(name);
105     }
106
107     // Override the eat() method
108     @Override
109     void eat() {
110         System.out.println(name + " is eating fish.");
111     }
112
113     void meow() {
114         System.out.println(name + " is meowing.");
115     }
116 }
```

- 5. Abstraction:** Abstraction is the process of hiding the implementation details and showing only the functionality to the user.

```
117
118 // 5. Abstraction
119 // Abstraction is the process of hiding the implementation details and showing only the functionality to the user.
120 // It can be achieved through abstract classes and interfaces.
121
122 // Abstract class
123 abstract class Shape {
124     abstract double calculateArea();
125
126     void display() {
127         System.out.println("This is a shape.");
128     }
129 }
130
131 class Circle extends Shape {
132     double radius;
133
134     Circle(double radius) {
135         this.radius = radius;
136     }
137
138     @Override
139     double calculateArea() {
140         return Math.PI * radius * radius;
141     }
142 }
```

6. **Interfaces**: An interface is a completely abstract class that contains only abstract methods.

```
145 // 6. Interfaces
146 // An interface is a completely abstract class that contains only abstract methods.
147 // It can be used to achieve multiple inheritance in Java.
148
149 interface Drawable {
150     void draw();
151 }
152
153 class Rectangle extends Shape implements Drawable {
154     double length;
155     double width;
156
157     Rectangle(double length, double width) {
158         this.length = length;
159         this.width = width;
160     }
161
162     @Override
163     double calculateArea() {
164         return length * width;
165     }
166
167     @Override
168     public void draw() {
169         System.out.println("Drawing a rectangle");
170     }
171 }
```

```

172 // Main class to demonstrate all concepts
173 public class P12JavaOOPConceptsForBeginners {
174     Run|Debug
175     public static void main(String[] args) {
176         // 1. Classes and Objects
177         System.out.println("1. Classes and Objects:");
178         Car myCar = new Car(brand:"Toyota", model:"Corolla", year:2022);
179         myCar.displayInfo();
180
181         // 2. Encapsulation
182         System.out.println("\n2. Encapsulation:");
183         BankAccount account = new BankAccount(accountNumber:"123456", initialBalance:1000);
184         account.deposit(amount:500);
185         account.withdraw(amount:200);
186         System.out.println("Account balance: $" + account.getBalance());
187
188         // 3. Inheritance
189         System.out.println("\n3. Inheritance:");
190         Dog myDog = new Dog(name:"Buddy");
191         myDog.eat();
192         myDog.bark();
193
194         // 4. Polymorphism
195         System.out.println("\n4. Polymorphism:");
196         Animal myAnimal = new Cat(name:"Whiskers");
197         myAnimal.eat(); // This will call the overridden method in Cat
198
199         // 5. Abstraction
200         System.out.println("\n5. Abstraction:");
201         Shape myCircle = new Circle(radius:5);
202         System.out.println("Circle area: " + myCircle.calculateArea());
203
204         // 6. Interfaces
205         System.out.println("\n6. Interfaces:");
206         Rectangle myRectangle = new Rectangle(length:4, width:5);
207         myRectangle.draw();
208         System.out.println("Rectangle area: " + myRectangle.calculateArea());
209     }

```

```

1. Classes and Objects:
Car: 2022 Toyota Corolla

2. Encapsulation:
Account balance: $1300.0

3. Inheritance:
Buddy is eating.
Buddy is barking.

4. Polymorphism:
Whiskers is eating fish.

5. Abstraction:
Circle area: 78.53981633974483

6. Interfaces:
Drawing a rectangle
Rectangle area: 20.0
PS C:\Codebase\Java\Code>

```

## Composition:

Composition is the mechanism that allows for the "objects can be made up of other objects" characteristic. Composition allows us to express a has-a relationship. It allows us to reuse code by creating fields in our objects that are other objects. We then delegate to the contained object when we want to take advantage of the capabilities of that object.

### Program 13 - Demonstrating the use of Composition

```
J P13ComputerSystemComposition.java > P13ComputerSystemComposition
1 // Computer System Composition Example
2
3 // Component classes
4 class CPU {
5     private String model;
6     private double clockSpeed;
7
8     public CPU(String model, double clockSpeed) {
9         this.model = model;
10        this.clockSpeed = clockSpeed;
11    }
12
13    public void processData() {
14        System.out.println("CPU " + model + " processing data at " + clockSpeed + " GHz");
15    }
16 }
17
18 class RAM {
19     private String type;
20     private int capacityGB;
21
22     public RAM(String type, int capacityGB) {
23         this.type = type;
24         this.capacityGB = capacityGB;
25     }
26
27     public void loadData() {
28         System.out.println("Loading data into " + capacityGB + "GB " + type + " RAM");
29     }
30 }
31
32 class Storage {
33     private String type;
34     private int capacityGB;
35
36     public Storage(String type, int capacityGB) {
37         this.type = type;
38         this.capacityGB = capacityGB;
39     }
40
41     public void storeData() {
42         System.out.println("Storing data on " + capacityGB + "GB " + type);
43     }
44 }
45
46 class GPU {
47     private String model;
48     private int memoryGB;
49
50     public GPU(String model, int memoryGB) {
51         this.model = model;
```

```

46 class GPU {
50     public GPU(String model, int memoryGB) {
52         this.memoryGB = memoryGB;
53     }
54
55     public void renderGraphics() {
56         System.out.println("Rendering graphics using " + model + " with " + memoryGB + "GB memory");
57     }
58 }
59
60 // Main class using composition
61 class Computer {
62     private CPU cpu;
63     private RAM ram;
64     private Storage storage;
65     private GPU gpu;
66
67     public Computer(CPU cpu, RAM ram, Storage storage, GPU gpu) {
68         this.cpu = cpu;
69         this.ram = ram;
70         this.storage = storage;
71         this.gpu = gpu;
72     }
73
74     public void bootUp() {
75         System.out.println("Booting up the computer...");
76         cpu.processData();
77         ram.loadData();
78         storage.storeData();
79         gpu.renderGraphics();
80         System.out.println("Computer is ready to use!");
81     }
82
83     public void shutDown() {
84         System.out.println("Shutting down the computer...");
85         storage.storeData(); // Saving data before shutdown
86         System.out.println("Computer has been shut down.");
87     }
88
89     // Getter methods for components
90     public CPU getCpu() { return cpu; }
91     public RAM getRam() { return ram; }
92     public Storage getStorage() { return storage; }
93     public GPU getGpu() { return gpu; }
94 }
95
96 // Example usage
97 public class P13ComputerSystemComposition {
98     Run|Debug
99     public static void main(String[] args) {

```

```

97  public class P13ComputerSystemComposition {
98      Run | Debug
99      public static void main(String[] args) {
100         CPU cpu = new CPU(model:"Intel i7", clockSpeed:3.6);
101         RAM ram = new RAM(type:"DDR4", capacityGB:16);
102         Storage storage = new Storage(type:"SSD", capacityGB:512);
103         GPU gpu = new GPU(model:"NVIDIA RTX 3070", memoryGB:8);
104
105         Computer myComputer = new Computer(cpu, ram, storage, gpu);
106
107         myComputer.bootUp();
108         System.out.println("Performing some operations...");
109         myComputer.getCpu().processData();
110         myComputer.getGpu().renderGraphics();
111         System.out.println();
112         myComputer.shutDown();
113     }
114

```

PROBLEMS 36 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

1f401\bin' 'P13ComputerSystemComposition'
Booting up the computer...
CPU Intel i7 processing data at 3.6 GHz
Loading data into 16GB DDR4 RAM
Storing data on 512GB SSD
Rendering graphics using NVIDIA RTX 3070 with 8GB memory
Computer is ready to use!

Performing some operations...
CPU Intel i7 processing data at 3.6 GHz
Rendering graphics using NVIDIA RTX 3070 with 8GB memory

Shutting down the computer...
Storing data on 512GB SSD
Computer has been shut down.
PS C:\Codebase\Java\Code>

```

## Abstract Base Class:

- Like an interface, an abstract class cannot be instantiated — only subclasses of an abstract class can be instantiated.

- Like an interface, an abstract base class can define **methods** (definition only — no implementation) and then force subclasses to provide an implementation.
- Like a regular base class, an abstract base class can provide fully-implemented methods that get inherited by child classes.
- Like a regular base class, an abstract base class can contain fields that get inherited by child classes.

## Interface:

An interface is a user-defined type, just like classes and enums. The big difference is that interfaces are never instantiated.

Declaring an Interface:

Declaring an interface is similar to declaring a class. Just as with a class, the interface must reside in a file with the same name as the interface and a **.java** extension. For example, the following interface must reside in a file called **Vehicle.java**:

```
public interface Vehicle {  
    public void moveForward(int milesPerHour);  
    public void moveBackward(int milesPerHour);  
    public void stop();  
    public void turnLeft();  
    public void turnRight();  
    public void engineOn();  
    public void engineOff();  
}
```

```
J P14AbstractAndInterface.java > Vehicle
1 // Abstract class and interface examples
2
3 // Example 1: Shape hierarchy
4 abstract class Shape {
5     protected String color;
6
7     public Shape(String color) {
8         this.color = color;
9     }
10
11    // Abstract method
12    public abstract double calculateArea();
13
14    // Concrete method
15    public void displayColor() {
16        System.out.println("Color: " + color);
17    }
18}
19
20 class Circle extends Shape {
21     private double radius;
22
23     public Circle(String color, double radius) {
24         super(color);
25         this.radius = radius;
26     }
27
28     @Override
29     public double calculateArea() {
30         return Math.PI * radius * radius;
31     }
32 }
33
34 class Rectangle extends Shape {
35     private double length;
36     private double width;
37
38     public Rectangle(String color, double length, double width) {
39         super(color);
40         this.length = length;
41         this.width = width;
42     }
43
44     @Override
45     public double calculateArea() {
46         return length * width;
47     }
48 }
49
50 // Example 2: Vehicle interface and implementations
51 interface Vehicle {
52     void start();
53     void stop();
54     int getFuelLevel();
55 }
```

```

57 abstract class AbstractVehicle implements Vehicle {
58     protected int fuelLevel;
59
60     public AbstractVehicle(int initialFuel) {
61         this.fuelLevel = initialFuel;
62     }
63
64     @Override
65     public int getFuelLevel() {
66         return fuelLevel;
67     }
68
69     // Abstract method specific to AbstractVehicle
70     public abstract void refuel(int amount);
71 }
72
73 class Car extends AbstractVehicle {
74     public Car(int initialFuel) {
75         super(initialFuel);
76     }
77
78     @Override
79     public void start() {
80         System.out.println("Car engine started");
81     }
82
83     @Override
84     public void stop() {
85         System.out.println("Car engine stopped");
86     }
87
88     @Override
89     public void refuel(int amount) {
90         fuelLevel += amount;
91         System.out.println("Car refueled. New fuel level: " + fuelLevel);
92     }
93 }
94
95 class Bicycle implements Vehicle {
96     @Override
97     public void start() {
98         System.out.println("Bicycle started moving");
99     }
100
101    @Override
102    public void stop() {
103        System.out.println("Bicycle stopped moving");
104    }
105
106    @Override
107    public int getFuelLevel() {
108        return 100; // Bicycles don't use fuel, so always return 100%
109    }
110 }
```

```
112 // Example 3: Payment processing
113 interface PaymentProcessor {
114     boolean processPayment(double amount);
115     void refundPayment(double amount);
116 }
117
118 abstract class OnlinePaymentProcessor implements PaymentProcessor {
119     protected String merchantId;
120
121     public OnlinePaymentProcessor(String merchantId) {
122         this.merchantId = merchantId;
123     }
124
125     // Abstract method specific to OnlinePaymentProcessor
126     public abstract boolean verifyMerchant();
127 }
128
129 class CreditCardProcessor extends OnlinePaymentProcessor {
130     public CreditCardProcessor(String merchantId) {
131         super(merchantId);
132     }
133
134     @Override
135     public boolean processPayment(double amount) {
136         if (verifyMerchant()) {
137             System.out.println("Processing credit card payment of $" + amount);
138             return true;
139         }
140         return false;
141     }
142
143     @Override
144     public void refundPayment(double amount) {
145         System.out.println("Refunding credit card payment of $" + amount);
146     }
147
148     @Override
149     public boolean verifyMerchant() {
150         // Simulate merchant verification
151         return merchantId.startsWith(prefix:"CC");
152     }
153 }
154
155 class PayPalProcessor extends OnlinePaymentProcessor {
156     public PayPalProcessor(String merchantId) {
157         super(merchantId);
158     }
159
160     @Override
161     public boolean processPayment(double amount) {
162         if (verifyMerchant()) {
163             System.out.println("Processing PayPal payment of $" + amount);
164             return true;
165         }
166         return false;
167     }
168 }
```

```
155 class PayPalProcessor extends OnlinePaymentProcessor {
156     @Override
157     public void refundPayment(double amount) {
158         System.out.println("Refunding PayPal payment of $" + amount);
159     }
160
161     @Override
162     public boolean verifyMerchant() {
163         // Simulate merchant verification
164         return merchantId.startsWith(prefix:"PP");
165     }
166 }
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181 public class P14AbstractAndInterface {
182     Run | Debug
183     public static void main(String[] args) {
184         // Example 1: Shape hierarchy
185         System.out.println("Example 1: Shape Hierarchy");
186         Shape circle = new Circle(color:"Red", radius:5);
187         Shape rectangle = new Rectangle(color:"Blue", length:4, width:6);
188
189         System.out.println("Circle:");
190         circle.displayColor();
191         System.out.println("Area: " + circle.calculateArea());
192
193         System.out.println("\nRectangle:");
194         rectangle.displayColor();
195         System.out.println("Area: " + rectangle.calculateArea());
196
197         // Example 2: Vehicle interface and implementations
198         System.out.println("\nExample 2: Vehicle Interface");
199         Vehicle car = new Car(initialFuel:50);
200         Vehicle bicycle = new Bicycle();
201
202         car.start();
203         System.out.println("Car fuel level: " + car.getFuelLevel());
204         ((Car) car).refuel(amount:20);
205         car.stop();
206
207         bicycle.start();
208         System.out.println("Bicycle 'fuel' level: " + bicycle.getFuelLevel());
209         bicycle.stop();
210
211         // Example 3: Payment processing
212         System.out.println("\nExample 3: Payment Processing");
213         PaymentProcessor creditCard = new CreditCardProcessor(merchantId:"CC123");
214         PaymentProcessor paypal = new PayPalProcessor(merchantId:"PP456");
215
216         creditCard.processPayment(amount:100.50);
217         creditCard.refundPayment(amount:20.00);
218
219         paypal.processPayment(amount:75.25);
220         paypal.refundPayment(amount:75.25);
221 }
```

## Program 15 - Demonstrating of Parameterized Constructor

```
class P15ParameterizedConstructor {  
    // Instance variables  
    String name;  
    int age;  
  
    // Parameterized constructor  
    public P15ParameterizedConstructor(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    // Method to display the values  
    public void display() {  
        System.out.println("Name: " + name);  
        System.out.println("Age: " + age);  
    }  
  
    public static void main(String[] args) {  
        // Creating an object using the parameterized constructor  
        P15ParameterizedConstructor person = new P15ParameterizedConstructor("Anjali", 25);  
        person.display();  
    }  
}
```

### Access Modifiers:

Modifier	Class Access?	Package Access?	Subclass Access?	World Access?
<b>public</b>	Yes	Yes	Yes	Yes
<b>protected</b>	Yes	Yes	Yes	No
<b>no modifier</b>	Yes	Yes	No	No
<b>private</b>	Yes	No	No	No

## Program 16 - Demonstrating use of Access Modifiers

```
class P16AccessModifiers {
    // Public variable: accessible from anywhere
    public String publicVar = "I am Public";

    // Private variable: accessible only within this class
    private String privateVar = "I am Private";

    // Protected variable: accessible within the same package and subclasses
    protected String protectedVar = "I am Protected";

    // Default (no modifier) variable: accessible within the same package
    String defaultVar = "I am Default";

    // Public method to access private variable
    public void displayPrivateVar() {
        System.out.println(privateVar);
    }

    public static void main(String[] args) {
        P16AccessModifiers obj = new P16AccessModifiers();

        // Accessing all variables
        System.out.println(obj.publicVar);      // Accessible anywhere
        System.out.println(obj.protectedVar);   // Accessible within the same package
        System.out.println(obj.defaultVar);     // Accessible within the same package
        obj.displayPrivateVar();              // Access private variable via public method
    }
}
```

## Simple File I/O:

Writing to a File: We will use a PrintWriter object to write to our files.

Reading From a File: We will use the Scanner object to read from our files.

## Program 17 - Demonstrating the use of various I/O Files Operations in Java

```
J P17JavaIO.java > ↗ P17JavaIO
1 import java.io.*;
2 import java.nio.file.*;
3 import java.util.Scanner;
4 import java.util.stream.Stream;
5
6 public class P17JavaIO {
7
8     Run | Debug
9     public static void main(String[] args) {
10         String fileName = "example.txt";
11         String content = "Hello, Java I/O!";
12
13         // Writing to a file
14         writeFile(fileName, content);
15
16         // Reading from a file using various methods
17         System.out.println("1. Using FileReader and BufferedReader:");
18         readWithBufferedReader(fileName);
19
20         System.out.println("\n2. Using FileInputStream:");
21         readWithFileInputStream(fileName);
22
23         System.out.println("\n3. Using Scanner:");
24         readWithScanner(fileName);
25
26         System.out.println("\n4. Using Files.readAllLines (Java NIO):");
27         readWithFilesReadAllLines(fileName);
28
29         System.out.println("\n5. Using Files.lines (Java NIO) with Stream API:");
30         readWithFilesLines(fileName);
31
32         // Demonstrating file operations
33         fileOperations(fileName);
34     }
35
36     public static void writeFile(String fileName, String content) {
37         try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {
38             writer.write(content);
39             System.out.println("Content written to file successfully.");
40         } catch (IOException e) {
41             System.err.println("Error writing to file: " + e.getMessage());
42         }
43     }
44
45     public static void readWithBufferedReader(String fileName) {
46         try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
47             String line;
48             while ((line = reader.readLine()) != null) {
49                 System.out.println(line);
50             }
51         } catch (IOException e) {
52             System.err.println("Error reading file: " + e.getMessage());
53         }
54     }
55 }
```

```

55     public static void readWithFileInputStream(String fileName) {
56         try (FileInputStream fis = new FileInputStream(fileName)) {
57             int content;
58             while ((content = fis.read()) != -1) {
59                 System.out.print((char) content);
60             }
61             System.out.println();
62         } catch (IOException e) {
63             System.err.println("Error reading file: " + e.getMessage());
64         }
65     }
66
67     public static void readWithScanner(String fileName) {
68         try (Scanner scanner = new Scanner(new File(fileName))) {
69             while (scanner.hasNextLine()) {
70                 System.out.println(scanner.nextLine());
71             }
72         } catch (FileNotFoundException e) {
73             System.err.println("File not found: " + e.getMessage());
74         }
75     }
76
77     public static void readWithFilesReadAllLines(String fileName) {
78         try {
79             Files.readAllLines(Paths.get(fileName)).forEach(System.out::println);
80         } catch (IOException e) {
81             System.err.println("Error reading file: " + e.getMessage());
82         }
83     }
84
85     public static void readWithFilesLines(String fileName) {
86         try (Stream<String> stream = Files.lines(Paths.get(fileName))) {
87             stream.forEach(System.out::println);
88         } catch (IOException e) {
89             System.err.println("Error reading file: " + e.getMessage());
90         }
91     }
92
93     public static void fileOperations(String fileName) {
94         File file = new File(fileName);
95
96         System.out.println("\nFile Operations:");
97         System.out.println("Exists: " + file.exists());
98         System.out.println("Is File: " + file.isFile());
99         System.out.println("Is Directory: " + file.isDirectory());
100        System.out.println("Can Read: " + file.canRead());
101        System.out.println("Can Write: " + file.canWrite());
102        System.out.println("Absolute Path: " + file.getAbsolutePath());
103
104        // Rename the file
105        File newFile = new File("new_" + fileName);
106        if (file.renameTo(newFile)) {
107            System.out.println("File renamed successfully.");
108        } else {

```

```
103
104     // Rename the file
105     File newFile = new File("new_" + fileName);
106     if (file.renameTo(newFile)) {
107         System.out.println("File renamed successfully.");
108     } else {
109         System.out.println("Failed to rename the file.");
110     }
111
112     // Delete the file
113     if (newFile.delete()) {
114         System.out.println("File deleted successfully.");
115     } else {
116         System.out.println("Failed to delete the file.");
117     }
118 }
119 }
```

PROBLEMS 22 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P17JavaIO'
Content written to file successfully.
1. Using FileReader and BufferedReader:
Hello, Java I/O!

2. Using FileInputStream:
Hello, Java I/O!

3. Using Scanner:
Hello, Java I/O!

4. Using Files.readAllLines (Java NIO):
Hello, Java I/O!

5. Using Files.lines (Java NIO) with Stream API:
Hello, Java I/O!

File Operations:
Exists: true
Is File: true
Is Directory: false
Can Read: true
Can Write: true
Absolute Path: C:\Codebase\Java\Code\example.txt
File renamed successfully.
File deleted successfully.
PS C:\Codebase\Java\Code>
```

## Exception Handling in Java:

Exceptions:

Exceptions are events that occur during the execution of a program and disrupt its normal flow. They are typically errors or unexpected conditions that require special handling. Exceptions in Java are categorized into:

- Checked Exceptions: These are exceptions that are checked at compile time, and the programmer must handle them (e.g., `IOException`).
- Unchecked Exceptions: These are exceptions that occur at runtime due to programming errors (e.g., `ArithmaticException`, `NullPointerException`).
- Errors: These are serious problems that applications should not attempt to handle (e.g., `OutOfMemoryError`).

Try-Catch Block:

The `try` block contains code that might throw an exception, while the `catch` block is used to handle that exception. This mechanism prevents the program from terminating unexpectedly and allows it to deal with runtime errors gracefully.

Finally Block:

The `finally` block contains code that is always executed, regardless of whether an exception was thrown or not. It is commonly used for cleanup tasks like closing resources (e.g., closing file streams or database connections).

Throw Keyword:

The `throw` keyword is used to explicitly throw an exception from a method or block of code. It is often used for custom error handling, allowing the programmer to signal an exceptional condition manually.

Throws Keyword:

The `throws` keyword is used in the method signature to declare that a method might throw one or more exceptions. This informs the calling method that it must handle the potential exception or propagate it further.

Custom Exceptions:

Java allows developers to create custom exceptions by extending the `Exception` class or `Runtimeexception` class. This is useful when the standard exceptions do not adequately represent a specific error condition in the program.

Multiple Catch Blocks:

A `try` block can be followed by multiple `catch` blocks to handle different types of exceptions separately. Each `catch` block is designed to handle a specific exception type.

Multi-Catch Block (Java 7+):

Java 7 introduced the ability to catch multiple exceptions in a single `catch` block using the pipe

(|) symbol. This allows you to handle different exceptions in a single catch clause, making the code more concise.

### Exception Propagation:

When an exception occurs, Java propagates the exception up the call stack to find a matching **catch** block. If no handler is found, the program terminates. Exception propagation allows exceptions to be caught and handled at higher levels of the method call stack.

### Common Exception Classes:

Java provides several built-in exception classes that cover most common error conditions. These include:

- **ArithmaticException**: Thrown when an illegal arithmetic operation, such as division by zero, is performed.
- **NullPointerException**: Occurs when trying to use an object reference that is **null**.
- **ArrayIndexOutOfBoundsException**: Raised when an invalid index is accessed in an array.
- **FileNotFoundException**: Thrown when attempting to access a file that does not exist.

## Program 18 - Demonstrating the whole concept of Exception Handling

```
J P18BankAccountExceptionDemo.java > P18BankAccountExceptionDemo
1 import java.util.Scanner;
2
3 // Custom checked exception
4 class InsufficientFundsException extends Exception {
5     private double amount;
6
7     public InsufficientFundsException(double amount) {
8         super("Insufficient funds. You need " + amount + " more to complete this transaction.");
9         this.amount = amount;
10    }
11
12    public double getAmount() {
13        return amount;
14    }
15 }
16
17 // Custom unchecked exception
18 class InvalidAccountIdException extends RuntimeException {
19     private String accountId;
20
21     public InvalidAccountIdException(String accountId) {
22         super("Invalid account ID: " + accountId);
23         this.accountId = accountId;
24     }
25
26     public String getAccountId() {
27         return accountId;
28     }
29 }
30
31 // Main class demonstrating exception handling
32 public class P18BankAccountExceptionDemo {
33
34     Run|Debug
35     public static void main(String[] args) {
36         BankAccount account = new BankAccount(accountId:"12345", initialBalance:1000.0);
37         Scanner scanner = new Scanner(System.in);
38
39         while (true) {
40             try {
41                 // Demonstrate handling multiple types of exceptions
42                 System.out.print("Enter account ID: ");
43                 String accountId = scanner.nextLine();
44                 System.out.print("Enter amount to withdraw: ");
45                 double amount = Double.parseDouble(scanner.nextLine());
46
47                 // This may throw InvalidAccountIdException or InsufficientFundsException
48                 account.withdraw(accountId, amount);
49
50                 System.out.println("Withdrawal successful. New balance: " + account.getBalance());
51                 break;
52             } catch (InsufficientFundsException e) {
53                 // Handle the custom checked exception
54                 System.out.println("Error: " + e.getMessage());
55                 System.out.println("You need $" + e.getAmount() + " more. Would you like to try again? (y/n)");
56             }
57         }
58     }
59 }
```

```

32 public class P18BankAccountExceptionDemo {
33
34     public static void main(String[] args) {
35         try {
36             // Handle the custom unchecked exception
37             System.out.println("Error: " + e.getMessage());
38             System.out.println("Please try again with a valid account ID.");
39         } catch (NumberFormatException e) {
40             // Handle invalid input for withdrawal amount
41             System.out.println("Error: Invalid amount entered. Please enter a valid number.");
42         } catch (Exception e) {
43             // Generic exception handler for any unexpected exceptions
44             System.out.println("An unexpected error occurred: " + e.getMessage());
45             e.printStackTrace();
46             break;
47         } finally {
48             // This block always executes, regardless of whether an exception occurred
49             System.out.println("Transaction attempt completed.");
50         }
51     }
52
53     scanner.close();
54 }
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80 // Bank account class with methods that may throw exceptions
81 class BankAccount {
82     private String accountId;
83     private double balance;
84
85     public BankAccount(String accountId, double initialBalance) {
86         this.accountId = accountId;
87         this.balance = initialBalance;
88     }
89
90     public double getBalance() {
91         return balance;
92     }
93
94     // This method demonstrates throwing both custom checked and unchecked exceptions
95     public void withdraw(String accountId, double amount) throws InsufficientFundsException {
96         // Validate account ID (may throw unchecked exception)
97         if (!this.accountId.equals(accountId)) {
98             throw new InvalidAccountException(accountId);
99         }
100
101         // Check for sufficient funds (may throw checked exception)
102         if (amount > balance) {
103             double shortfall = amount - balance;
104             throw new InsufficientFundsException(shortfall);
105         }
106
107         // Perform withdrawal if all checks pass
108         balance -= amount;
109     }
110 }

```

# Day 5 - Caching, Collections and Threads Concepts

## Program 19 - Demonstrating the use of Enum

```
J P19CoffeShopEnumEg.java > P19CoffeShopEnumEg
1  import java.util.Scanner;
2
3  public class P19CoffeShopEnumEg {
4      // Enum definition for coffee sizes
5      public enum CoffeeSize {
6          SMALL(price:3.5, ounces:8),
7          MEDIUM(price:4.0, ounces:12),
8          LARGE(price:4.5, ounces:16),
9          EXTRA_LARGE(price:5.0, ounces:20);
10
11     private final double price;
12     private final int ounces;
13
14     // Constructor for the enum
15     CoffeeSize(double price, int ounces) {
16         this.price = price;
17         this.ounces = ounces;
18     }
19
20     // Getter methods
21     public double getPrice() {
22         return price;
23     }
24
25     public int getOunces() {
26         return ounces;
27     }
28 }
29
Run | Debug
30 public static void main(String[] args) {
31     Scanner scanner = new Scanner(System.in);
32
33     System.out.println("Welcome to our Coffee Shop!");
34     System.out.println("Available sizes:");
35
36     // Display all available coffee sizes
37     for (CoffeeSize size : CoffeeSize.values()) {
38         System.out.printf("%s (%d oz) - $%.2f\n",
39                         size.name(),
40                         size.getOunces(),
41                         size.getPrice());
42     }
43
44     System.out.print("Enter your choice (SMALL/MEDIUM/LARGE/EXTRA_LARGE): ");
45     String userChoice = scanner.nextLine().toUpperCase();
46
47     try {
48         CoffeeSize selectedSize = CoffeeSize.valueOf(userChoice);
49         System.out.printf("You've selected a %s coffee (%d oz).\n",
50                         selectedSize.name(),
51                         selectedSize.getOunces());
52         System.out.printf("Price: $%.2f\n", selectedSize.getPrice());
53     } catch (IllegalArgumentException e) {
54         System.out.println("Invalid size selected. Please try again.");
55     }
56
57     scanner.close();
58 }
59 }
60 }
```

```
Welcome to our Coffee Shop!
Available sizes:
SMALL (8 oz) - $3.50
MEDIUM (12 oz) - $4.00
LARGE (16 oz) - $4.50
EXTRA_LARGE (20 oz) - $5.00
Enter your choice (SMALL/MEDIUM/LARGE/EXTRA_LARGE): SMALL
You've selected a SMALL coffee (8 oz).
Price: $3.50
PS C:\Codebase\Java\Code> █
```

## Program 20 - Caching Concepts

```
J P20MovieRecommendationCache.java > P20MovieRecommendationCache > main(String[])
1 import java.util.LinkedHashMap;
2 import java.util.Map;
3
4 public class P20MovieRecommendationCache {
5     /**
6      * L1 Cache for storing recent movie recommendations.
7      *
8      * Characteristics:
9      * 1. Cache size: 100 entries.
10     * 2. Access-order iteration (LRU eviction).
11     * 3. When the cache exceeds the maximum capacity, the Least recently accessed entry is removed.
12     * 4. Load factor of 0.75 improves performance.
13     */
14    private static final int MAX_ENTRIES = 100;
15    private final Map<String, String> recentMovieCache =
16        new LinkedHashMap<String, String>(MAX_ENTRIES, 0.75f, true) {
17            @Override
18            protected boolean removeEldestEntry(Map.Entry<String, String> eldest) {
19                return size() > MAX_ENTRIES;
20            }
21        };
22
23    /**
24     * L2 Cache for storing popular movie recommendations.
25     *
26     * Characteristics:
27     * 1. Cache size: 1000 entries.
28     * 2. Access-order iteration (LRU eviction).
29     * 3. When the cache exceeds the maximum capacity, the Least recently accessed entry is removed.
30     * 4. Load factor of 0.75 improves performance.
31     */
32    private static final int MAX_ENTRIES_L2 = 1000;
33    private final Map<String, String> popularMovieCache =
34        new LinkedHashMap<String, String>(MAX_ENTRIES_L2, 0.75f, true) {
35            @Override
36            protected boolean removeEldestEntry(Map.Entry<String, String> eldest) {
37                return size() > MAX_ENTRIES_L2;
38            }
39        };
40
41    /**
42     * Adds a movie recommendation to the L1 (recent) cache.
43     */
44    public void addToRecentMovies(String movie, String recommendation) {
45        recentMovieCache.put(movie, recommendation);
46    }
47
48    /**
49     * Adds a movie recommendation to the L2 (popular) cache.
50     */
```

```
4  public class P20MovieRecommendationCache {
5  public void addToPopularMovies(String movie, String recommendation) {
6  }
7
8  /**
9   * Displays the contents of the recent movie cache.
10  */
11 public void displayRecentMovies() {
12     System.out.println("Recent Movie Cache:");
13     for (Map.Entry<String, String> entry : recentMovieCache.entrySet()) {
14         System.out.println(entry.getKey() + ": " + entry.getValue());
15     }
16 }
17
18 /**
19  * Displays the contents of the popular movie cache.
20  */
21 public void displayPopularMovies() {
22     System.out.println("Popular Movie Cache:");
23     for (Map.Entry<String, String> entry : popularMovieCache.entrySet()) {
24         System.out.println(entry.getKey() + ": " + entry.getValue());
25     }
26 }
27
28 Run|Debug
29 public static void main(String[] args) {
30     P20MovieRecommendationCache cache = new P20MovieRecommendationCache();
31
32     // Adding elements to recent movie cache (L1)
33     cache.addToRecentMovies(movie:"Inception", recommendation:"Mind-bending thriller");
34     cache.addToRecentMovies(movie:"The Matrix", recommendation:"Classic sci-fi");
35     cache.addToRecentMovies(movie:"Interstellar", recommendation:"Space exploration epic");
36
37     // Adding elements to popular movie cache (L2)
38     cache.addToPopularMovies(movie:"Titanic", recommendation:"Epic romance");
39     cache.addToPopularMovies(movie:"Avatar", recommendation:"Visually stunning adventure");
40     cache.addToPopularMovies(movie:"Avengers: Endgame", recommendation:"Superhero blockbuster");
41
42     // Display contents of both caches
43     cache.displayRecentMovies();
44     cache.displayPopularMovies();
45 }
```

PROBLEMS 21 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-d
Recent Movie Cache:
Inception: Mind-bending thriller
The Matrix: Classic sci-fi
Interstellar: Space exploration epic
Popular Movie Cache:
Titanic: Epic romance
Avatar: Visually stunning adventure
Avengers: Endgame: Superhero blockbuster
PS C:\Codebase\Java\Code>
```

```
Avengers: Endgame: Superhero blockbuster
Avengers: Endgame: Superhero blockbuster
PS C:\Codebase\Java\Code>
```

---

## Collections:

A collection is simply a data structure that groups multiple elements into a single unit.

### Commonly Used Collections:

---

#### List

List is an interface that extends Collection. It is an ordered collection of items that may include duplicate elements.

#### ArrayList

The **ArrayList** is the List implementation we will use the most. Like any implementation of the List interface, the ArrayList has all of the features of the Collection interface and all of the features of the List interface.

#### Stack

A **Stack** is another List implementation that adds in specialty **methods** to treat the List like a last-in-first-out (LIFO) stack. Those specialty **methods** are **push** to place an object on the top of the Stack, **pop** to remove an object from the top of the Stack, and **peek** to see what object is on the top of the Stack.

#### Set

Set is another interface that extends Collection. There are two differences between a List and a Set:

1. Each item in a Set must be unique: all duplicates are ignored. **Lists** do allow duplicate items.
2. A Set does not maintain order by default, whereas a List will. This means that when you retrieve items from a Set, the items may appear in a different order each time.

#### HashSet

A **HashSet** is the type of Set we typically use. It determines whether an object is unique or not based on the hashCode of the object, a number calculated based on information inside the object.

---

## TreeSet

A **TreeSet** is a Set that does maintain an order. It will default to the natural order of the objects, numerical or alphabetical order. If the objects are more complex you can provide a **Comparator** that tells the TreeSet how to compare and order the objects.

## Map

While not strictly part of the Collections Framework, a Map is an object that **maps** keys to values. We can retrieve Collections of the keys or values in a Map, so they are tied to the Collections Framework.

All keys in a Map must be unique, while values can be duplicated. Because of that when we get all the keys we get a Set; getting all the values gives us a List.

## HashMap

A **HashMap** is the Map we use most often. It determines if a key is unique based on the hashCode of that key. Entries in the HashMap have no defined order and can not guarantee that any order will remain consistent over time.

## TreeMap

A **TreeMap** is a Map that will maintain an order based on the keys. Similar to the TreeSet we talked about previously, it will use the natural order or a Comparator to determine the order.

---

## Program 21 - Demonstrating the concepts of Collections

```
J P21CollectionsConcepts.java > P21CollectionsConcepts
 1 import java.util.*;
 2 import java.util.concurrent.*;
 3 import java.util.stream.Collectors;
 4
 5 public class P21CollectionsConcepts {
 6
 7     Run|Debug
 8     public static void main(String[] args) {
 9         demonstrateList();
10         demonstrateSet();
11         demonstrateQueue();
12         demonstrateMap();
13         demonstrateUtilityClasses();
14     }
15
16     private static void demonstrateList() {
17         System.out.println("\n--- List Interfaces ---");
18
19         // ArrayList: Dynamic array implementation
20         List<String> arrayList = new ArrayList<>();
21         arrayList.add("Apple");
22         arrayList.add("Banana");
23         arrayList.add("Cherry");
24         System.out.println("ArrayList: " + arrayList);
25
26         // LinkedList: Doubly-Linked List implementation
27         List<String> linkedList = new LinkedList<>(arrayList);
28         linkedList.add(index:1, element:"Blueberry");
29         System.out.println("LinkedList: " + linkedList);
30
31         // Vector: Thread-safe dynamic array (Legacy class)
32         Vector<String> vector = new Vector<>(arrayList);
33         vector.add("Date");
34         System.out.println("Vector: " + vector);
35
36         // Stack: LIFO stack (extends Vector, Legacy class)
37         Stack<String> stack = new Stack<>();
38         stack.push(item:"Elderberry");
39         stack.push(item:"Fig");
40         System.out.println("Stack: " + stack);
41         System.out.println("Popped from stack: " + stack.pop());
42
43         // Operations
44         Collections.sort(arrayList);
45         System.out.println("Sorted ArrayList: " + arrayList);
46         System.out.println("Binary search for 'Cherry': " + Collections.binarySearch(arrayList, key:"Cherry"));
47     }
48
49     private static void demonstrateSet() {
50         System.out.println("\n--- Set Interfaces ---");
51
52         // HashSet: Hash table implementation, no guaranteed order
53         Set<String> hashSet = new HashSet<>();
54         hashSet.add("Red");
55         hashSet.add("Green");
56         hashSet.add("Blue");
57         hashSet.add("Red"); // Duplicate, won't be added
58         System.out.println("HashSet: " + hashSet);
59
60         // LinkedHashSet: Hash table and Linked List implementation, maintains insertion order
61         Set<String> linkedHashSet = new LinkedHashSet<>();
62         linkedHashSet.add("Dog");
63         linkedHashSet.add("Cat");
64         linkedHashSet.add("Bird");
65         System.out.println("LinkedHashSet: " + linkedHashSet);
66
67         // TreeSet: Red-black tree implementation, sorted order
```

```

46     private static void demonstrateSet() {
47         Set<String> treeSet = new TreeSet<>();
48         treeSet.add("Zebra");
49         treeSet.add("Elephant");
50         treeSet.add("Lion");
51         System.out.println("TreeSet: " + treeSet);
52
53         // Operations
54         Set<String> set1 = new HashSet<>(Arrays.asList("A", "B", "C"));
55         Set<String> set2 = new HashSet<>(Arrays.asList("B", "C", "D"));
56         set1.retainAll(set2); // Intersection
57         System.out.println("Intersection: " + set1);
58     }
59
60     private static void demonstrateQueue() {
61         System.out.println("\n--- Queue Interfaces ---");
62
63         // PriorityQueue: Priority heap implementation
64         Queue<Integer> priorityQueue = new PriorityQueue<>();
65         priorityQueue.offer(5);
66         priorityQueue.offer(1);
67         priorityQueue.offer(3);
68         System.out.println("PriorityQueue: " + priorityQueue);
69         System.out.println("Poll from PriorityQueue: " + priorityQueue.poll());
70
71         // LinkedList as Queue: FIFO queue
72         Queue<String> queue = new LinkedList<>();
73         queue.offer("First");
74         queue.offer("Second");
75         queue.offer("Third");
76         System.out.println("Queue: " + queue);
77         System.out.println("Poll from Queue: " + queue.poll());
78
79         // Deque: Double-ended queue
80         Deque<String> deque = new ArrayDeque<>();
81         deque.addFirst("Front");
82         deque.addLast("Back");
83         System.out.println("Deque: " + deque);
84         System.out.println("Poll first from Deque: " + deque.pollFirst());
85         System.out.println("Poll last from Deque: " + deque.pollLast());
86
87         // BlockingQueue: Thread-safe queue with blocking operations
88         BlockingQueue<String> blockingQueue = new LinkedBlockingQueue<>();
89         blockingQueue.offer("Task 1");
90         blockingQueue.offer("Task 2");
91         System.out.println("BlockingQueue: " + blockingQueue);
92     }
93
94     private static void demonstrateMap() {
95         System.out.println("\n--- Map Interfaces ---");
96
97         // HashMap: Hash table implementation
98         Map<String, Integer> hashMap = new HashMap<>();
99         hashMap.put("One", 1);
100        hashMap.put("Two", 2);
101        hashMap.put("Three", 3);
102        System.out.println("HashMap: " + hashMap);
103
104        // LinkedHashMap: Hash table and Linked List implementation, maintains insertion order
105        Map<String, String> linkedHashMap = new LinkedHashMap<>();
106        linkedHashMap.put("USA", "Washington D.C.");
107        linkedHashMap.put("UK", "London");
108        linkedHashMap.put("Japan", "Tokyo");
109        System.out.println("LinkedHashMap: " + linkedHashMap);
110
111        // TreeMap: Red-black tree implementation, sorted order
112    }

```

```

130
131     // TreeMap: Red-black tree implementation, sorted order
132     Map<String, Double> treeMap = new TreeMap<>();
133     treeMap.put(key:"Pi", value:3.14159);
134     treeMap.put(key:"Phi", value:1.61803);
135     treeMap.put(key:"e", value:2.71828);
136     System.out.println("TreeMap: " + treeMap);
137
138     // Hashtable: Thread-safe hash table (Legacy class)
139     Hashtable<Integer, String> hashtable = new Hashtable<>();
140     hashtable.put(key:1, value:"First");
141     hashtable.put(key:2, value:"Second");
142     System.out.println("Hashtable: " + hashtable);
143
144     // ConcurrentHashMap: Thread-safe hash table, better performance than Hashtable
145     ConcurrentHashMap<String, Integer> concurrentMap = new ConcurrentHashMap<>();
146     concurrentMap.put(key:"Concurrent", value:1);
147     concurrentMap.put(key:"Hash", value:2);
148     concurrentMap.put(key:"Map", value:3);
149     System.out.println("ConcurrentHashMap: " + concurrentMap);
150
151     // Operations
152     System.out.println("Keys in HashMap: " + hashMap.keySet());
153     System.out.println("Values in HashMap: " + hashMap.values());
154     System.out.println("Entries in HashMap: " + hashMap.entrySet());
155 }
156
157 private static void demonstrateUtilityClasses() {
158     System.out.println("\n--- Utility Classes ---");
159
160     // Arrays utility class
161     int[] numbers = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
162     Arrays.sort(numbers);
163     System.out.println("Sorted array: " + Arrays.toString(numbers));
164     System.out.println("Binary search for 5: " + Arrays.binarySearch(numbers, key:5));
165
166     // Collections utility class
167     List<String> list = Arrays.asList(...a:"apple", "banana", "cherry");
168     Collections.reverse(list);
169     System.out.println("Reversed list: " + list);
170     Collections.shuffle(list);
171     System.out.println("Shuffled list: " + list);
172     System.out.println("Max element: " + Collections.max(list));
173     System.out.println("Min element: " + Collections.min(list));
174
175     // Unmodifiable collections
176     List<String> unmodifiableList = Collections.unmodifiableList(new ArrayList<>(list));
177     // unmodifiableList.add("date"); // This would throw UnsupportedOperationException
178
179     // Synchronized collections
180     List<String> synchronizedList = Collections.synchronizedList(new ArrayList<>(list));
181     System.out.println("Synchronized list: " + synchronizedList);
182
183     // Java 8 Streams
184     List<Integer> numbers2 = Arrays.asList(...a:1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
185     List<Integer> evenSquares = numbers2.stream()
186         .filter(n -> n % 2 == 0)
187         .map(n -> n * n)
188         .collect(Collectors.toList());
189     System.out.println("Even squares: " + evenSquares);
190 }
191
192 }
```

```
--- List Interfaces ---
ArrayList: [Apple, Banana, Cherry]
LinkedList: [Apple, Blueberry, Banana, Cherry]
Vector: [Apple, Banana, Cherry, Date]
Stack: [Elderberry, Fig]
Popped from stack: Fig
Sorted ArrayList: [Apple, Banana, Cherry]
Binary search for 'Cherry': 2

--- Set Interfaces ---
HashSet: [Red, Blue, Green]
LinkedHashSet: [Dog, Cat, Bird]
TreeSet: [Elephant, Lion, Zebra]
Intersection: [B, C]

--- Queue Interfaces ---
PriorityQueue: [1, 5, 3]
Poll from PriorityQueue: 1
Queue: [First, Second, Third]
Poll from Queue: First
Deque: [Front, Back]
Poll first from Deque: Front
Poll last from Deque: Back
BlockingQueue: [Task 1, Task 2]

--- Map Interfaces ---
HashMap: {One=1, Two=2, Three=3}
LinkedHashMap: {USA=Washington D.C., UK=London, Japan=Tokyo}
TreeMap: {Phi=1.61803, Pi=3.14159, e=2.71828}
Hashtable: {2=Second, 1=First}
ConcurrentHashMap: {Concurrent=1, Hash=2, Map=3}
Keys in HashMap: [One, Two, Three]
Values in HashMap: [1, 2, 3]
Entries in HashMap: [One=1, Two=2, Three=3]

--- Utility Classes ---
Sorted array: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
Binary search for 5: 8
Reversed list: [cherry, banana, apple]
Shuffled list: [cherry, apple, banana]
Max element: cherry
Min element: apple
Synchronized list: [cherry, apple, banana]
Even squares: [4, 16, 36, 64, 100]
Shuffled list: [cherry, apple, banana]
Max element: cherry
Min element: apple
Synchronized list: [cherry, apple, banana]
Even squares: [4, 16, 36, 64, 100]
Synchronized list: [cherry, apple, banana]
Even squares: [4, 16, 36, 64, 100]
PS C:\Codebase\Java\Code>
```

---

## Threads Concepts:

### 1. Thread:

A thread is the smallest unit of a process. In Java, threads allow concurrent execution of two or more parts of a program, improving performance by utilizing CPU more efficiently. Each thread runs independently, although they share the process's memory space.

### 2. Creating Threads:

In Java, there are two ways to create a thread:

- Extending **Thread** Class: A class can extend the **Thread** class and override its **run()** method to define the thread's task.
- Implementing **Runnable** Interface: A class can implement the **Runnable** interface and define the **run()** method. This is the preferred way, as it allows the class to inherit from other classes as well.

### 3. Life Cycle of a Thread:

A thread has the following states:

- New: The thread is created but not yet started.
- Runnable: The thread is ready to run and waiting for CPU time.
- Running: The thread is actively executing.
- Blocked/Waiting: The thread is waiting for a resource or event (e.g., waiting for I/O).
- Terminated: The thread has finished executing.

### 4. Thread Methods:

- **start()**: Starts the execution of the thread by calling its **run()** method.
- **run()**: The method containing the code to be executed by the thread.
- **sleep()**: Pauses the execution of the current thread for a specified period.
- **join()**: Forces one thread to wait until another thread completes.
- **yield()**: Pauses the current thread to allow other threads of the same priority to execute.
- **interrupt()**: Interrupts a thread if it is in a waiting or sleeping state.

---

## 5. Thread Priority:

Each thread has a priority that influences the order in which it is scheduled to run. Java threads have priorities ranging from `MIN_PRIORITY` (1) to `MAX_PRIORITY` (10), with `NORM_PRIORITY` (5) as the default. Higher-priority threads are executed before lower-priority threads.

## 6. Synchronization:

When multiple threads try to access a shared resource, data inconsistency can occur. Synchronization ensures that only one thread can access a resource at a time to avoid conflicts. It is implemented using the `synchronized` keyword, which locks the resource for exclusive use by the thread.

## 7. Inter-Thread Communication:

Threads can communicate with each other using methods like `wait()`, `notify()`, and `notifyAll()`. These methods are used to make threads wait for certain conditions or signal each other when an event has occurred.

## 8. Daemon Threads:

Daemon threads are low-priority threads that run in the background and do not prevent the JVM from exiting. Examples include garbage collector threads. A thread can be set as a daemon using `setDaemon(true)`.

## 9. Concurrency Issues:

When multiple threads access shared resources simultaneously, issues like race conditions, deadlock, and thread interference can occur. These issues arise when threads conflict over resource access and must be carefully handled using synchronization or thread-safe mechanisms.

## 10. Thread Pool:

A thread pool is a group of pre-created threads that are reused to execute tasks. Instead of creating a new thread every time a task needs to be executed, a thread pool improves performance by reusing existing threads. Java provides `ExecutorService` to manage thread pools.

## Program 22 - Demonstrating Threads Concepts

```
1  public class P22ThreadingConcepts {
2      // volatile keyword ensures that the variable is always read from main memory
3      private volatile boolean flag = false;
4
5      // synchronized method to demonstrate thread synchronization
6      public synchronized void synchronizedMethod() {
7          System.out.println(Thread.currentThread().getName() + " entered synchronized method");
8          try {
9              Thread.sleep(1000);
10         } catch (InterruptedException e) {
11             e.printStackTrace();
12         }
13         System.out.println(Thread.currentThread().getName() + " exiting synchronized method");
14     }
15
16     // Runnable interface implementation
17     class MyRunnable implements Runnable {
18         public void run() {
19             System.out.println(Thread.currentThread().getName() + " is running");
20         }
21     }
22
23     // Thread class extension
24     class MyThread extends Thread {
25         public void run() {
26             System.out.println(Thread.currentThread().getName() + " is running");
27         }
28     }
29
30     public void demonstrateThreading() {
31         // Creating and starting a thread using Runnable
32         Thread thread1 = new Thread(new MyRunnable(), name:"RunnableThread");
33         thread1.start();
34
35         // Creating and starting a thread by extending Thread class
36         MyThread thread2 = new MyThread();
37         thread2.setName(name:"ExtendedThread");
38         thread2.start();
39
40         // Creating a thread using Lambda expression (Java 8+)
41         Thread thread3 = new Thread(() -> System.out.println(Thread.currentThread().getName() + " is running"), name:"LambdaThread");
42         thread3.start();
43
44         // Demonstrating thread states
45         System.out.println("Thread 3 state: " + thread3.getState());
46
47         // Thread priority
48         thread3.setPriority(Thread.MAX_PRIORITY);
49         System.out.println("Thread 3 priority: " + thread3.getPriority());
50
51         // Joining threads
52         try {
53             thread1.join();
54             thread2.join();
55             thread3.join();
56         } catch (InterruptedException e) {
57             e.printStackTrace();
58         }
59
60         // Demonstrating synchronized block
61         synchronized(this) {
62             System.out.println(Thread.currentThread().getName() + " in synchronized block");
63         }
64
65         // Using wait() and notify()
66         final Object lock = new Object();
67         Thread waiter = new Thread(() -> {
```

```

68     synchronized(lock) {
69         try {
70             System.out.println("Waiter is waiting");
71             lock.wait();
72             System.out.println("Waiter is notified");
73         } catch (InterruptedException e) {
74             e.printStackTrace();
75         }
76     }
77 });
78
79 Thread notifier = new Thread(() -> {
80     synchronized(lock) {
81         System.out.println("Notifier is notifying");
82         lock.notify();
83     }
84 });
85
86 waiter.start();
87 try {
88     Thread.sleep(millis:1000);
89 } catch (InterruptedException e) {
90     e.printStackTrace();
91 }
92 notifier.start();
93
94 // Demonstrating interrupt
95 Thread sleeper = new Thread(() -> {
96     try {
97         Thread.sleep(millis:5000);
98     } catch (InterruptedException e) {
99         System.out.println("Sleeper was interrupted");
100    }
101 });
102 sleeper.start();
103 sleeper.interrupt();
104
105 // Using ThreadLocal
106 ThreadLocal<Integer> threadLocal = new ThreadLocal<>();
107 threadLocal.set(value:42);
108 System.out.println("ThreadLocal value: " + threadLocal.get());
109
110 // Demonstrating deadlock (be cautious when running this)
111 final Object resource1 = new Object();
112 final Object resource2 = new Object();
113 Thread deadlockThread1 = new Thread(() -> {
114     synchronized(resource1) {
115         System.out.println("Thread 1: Holding resource 1...");
116         try { Thread.sleep(millis:100); } catch (InterruptedException e) {}
117         System.out.println("Thread 1: Waiting for resource 2...");
118         synchronized(resource2) {
119             System.out.println("Thread 1: Holding resource 1 and resource 2");
120         }
121     }
122 });
123 Thread deadlockThread2 = new Thread(() -> {
124     synchronized(resource2) {
125         System.out.println("Thread 2: Holding resource 2...");
126         try { Thread.sleep(millis:100); } catch (InterruptedException e) {}
127         System.out.println("Thread 2: Waiting for resource 1...");
128         synchronized(resource1) {
129             System.out.println("Thread 2: Holding resource 2 and resource 1");
130         }
131     }
132 });

```

```
132     });
133     deadlockThread1.start();
134     deadlockThread2.start();
135 }
136
Run | Debug
137 public static void main(String[] args) {
138     new P22ThreadingConcepts().demonstrateThreading();
139 }
140 }
141 Ctrl+L to chat, Ctrl+K to generate
```

PROBLEMS 21 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '1f401\bin\P22ThreadingConcepts'
RunnableThread is running
LambdaThread is running
ExtendedThread is running
Thread 3 state: RUNNABLE
Thread 3 priority: 5
main in synchronized block
Waiter is waiting
Notifier is notifying
Waiter is notified
ThreadLocal value: 42
Sleeper was interrupted
Thread 2: Holding resource 2...
Thread 1: Holding resource 1...
Thread 2: Waiting for resource 1...
Thread 1: Waiting for resource 2...
```

# Day 6 - Concepts of Caching

## Program 23 - Demonstrating Caching

```
J P23SimpleCache.java > ...
1 import java.util.HashMap;
2 import java.util.Map;
3
4 public class P23SimpleCache<K, V> {
5     private final Map<K, V> cache;
6
7     // Correct constructor name
8     public P23SimpleCache() {
9         this.cache = new HashMap<>();
10    }
11
12    public void put(K key, V value) {
13        cache.put(key, value);
14    }
15
16    public V get(K key) {
17        return cache.get(key);
18    }
19
20    public void remove(K key) {
21        cache.remove(key);
22    }
23
24    public void clear() {
25        cache.clear();
26    }
27
28    public int size() {
29        return cache.size();
30    }
31
32    Run | Debug
33    public static void main(String[] args) {
34        P23SimpleCache<String, String> cache = new P23SimpleCache<>();
35        cache.put(key:"key1", value:"value1");
36        cache.put(key:"key2", value:"value2");
37        cache.put(key:"key3", value:"value3");
38        System.out.println(cache.get(key:"key1"));
39        cache.remove(key:"key2");
40        System.out.println(cache.size());
41        cache.clear();
42        System.out.println(cache.size());
43    }
44
Ctrl+L to chat, Ctrl+K to generate
```

```
value1  
2  
0  
PS C:\Codebase\Java\Code> []
```

## Program 24 - Demonstrating of LRU Caching

The screenshot shows a Java code editor with a file named `LRUCache.java`. The code implements an LRU cache using a `LinkedHashMap`. It includes a constructor that sets a capacity of 3, and a `main` method that puts five key-value pairs into the cache and prints its state twice. The output in the terminal below shows the cache's internal state at different points in the execution.

```
JuntExceptionDemo.java 2 J P19CoffeShopEnumEg.java J P20MovieRecommendationCache.java  
J LRUCache.java > ...  
1 import java.util.LinkedHashMap;  
2 import java.util.Map;  
3  
4 public class LRUCache<K,V> extends LinkedHashMap<K,V> {  
5     private final int capacity;  
6  
7     public LRUCache(int capacity){  
8         super(capacity, loadFactor:0.75f, accessOrder:true);  
9         this.capacity = capacity;  
10    }  
11  
12    @Override  
13    protected boolean removeEldestEntry(Map.Entry<K,V> eldest){  
14        return size() > capacity;  
15    }  
16  
17    Run | Debug  
18    public static void main(String[] args) {  
19        LRUCache<String, String> cache = new LRUCache<>(capacity:3);  
20        cache.put(key:"key1", value:"value1");  
21        cache.put(key:"key2", value:"value2");  
22        cache.put(key:"key3", value:"value3");  
23        System.out.println(cache);  
24        cache.get(key:"key2");  
25        cache.put(key:"key4", value:"value4");  
26        cache.put(key:"key5", value:"value5");  
27        System.out.println(cache);  
28    }  
29  
30  
31  
32    }  
33  
PROBLEMS 37 OUTPUT DEBUG CONSOLE TERMINAL PORTS  
at LRUCache.main(P24LRUCache.java:20)  
PS C:\Codebase\Java\Code> ^C  
PS C:\Codebase\Java\Code>  
PS C:\Codebase\Java\Code> c:; cd 'c:\Codebase\Java\Code'; & 'C:\Program Files (x86)\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.java'  
{key1=value1, key2=value2, key3=value3}  
{key2=value2, key4=value4, key5=value5}  
PS C:\Codebase\Java\Code>
```

## Program 25 - Demonstrating Synchronization process using Bank Application

```
J P25NewBankDemo.java > P25NewBankDemo > main(String[])
1  class BankAccount {
2      private int balance = 0;
3
4      public BankAccount() {
5          balance = 0;
6      }
7
8      public synchronized void deposit(int amount) {
9          balance += amount;
10         System.out.println("Deposited: " + amount + " New Balance: " + balance);
11     }
12
13     public synchronized void withdraw(int amount) {
14         if (balance >= amount) {
15             balance -= amount;
16             System.out.println("Withdrawn: " + amount + " New Balance: " + balance);
17         } else {
18             System.out.println("Insufficient balance");
19         }
20     }
21
22     public int getBalance() {
23         return balance;
24     }
25 }
26
27 public class P25NewBankDemo {
28     Run|Debug
29     public static void main(String[] args) throws InterruptedException {
30         System.out.println("Bank Demo");
31
32         BankAccount account = new BankAccount();
33
34         Thread depositThread = new Thread(() -> {
35             for (int i = 0; i < 5; i++) {
36                 account.deposit(amount:100);
37                 try {
38                     Thread.sleep(millis:100);
39                 } catch (InterruptedException e) {
40                     e.printStackTrace();
41                 }
42             }, name:"Deposit Thread");
43
44         Thread withdrawThread = new Thread(() -> {
45             for (int i = 0; i < 5; i++) {
46                 account.withdraw(amount:80);
47                 try {
48                     Thread.sleep(millis:100);
49                 } catch (InterruptedException e) {
50                     e.printStackTrace();
51                 }
52             }, name:"Withdraw Thread");
53
54         depositThread.start();
55         withdrawThread.start();
56
57         depositThread.join();
58         withdrawThread.join();
59
60         System.out.println("Final Balance: " + account.getBalance());
61     }
62 }
63 }
64 }
```

## Program 26 - Demonstrating Caching

**NOTE:** In this Caching program we have to install a correct version of the Guava JAR ([guava-33.3.0-jre.jar](#)). It should be in the same directory as your `GuavaCacheExample.java` file.

```
J P26GuavaCacheExample.java > P26GuavaCacheExample > main(String[])
1 import java.util.Random;
2 import java.util.concurrent.ConcurrentHashMap;
3 import java.util.concurrent.TimeUnit;
4 import java.util.function.Function;
5 import java.util.concurrent.TimeUnit;
6
7 public class P26GuavaCacheExample {
8     Run|Debug
9     public static void main(String[] args) {
10         LoadingCache<String, String> cache = CacheBuilder.newBuilder()
11             .maximumSize(200000)
12             .expireAfterWrite(10, TimeUnit.MINUTES)
13             .build(new CacheLoader<String, String>() {
14                 @Override
15                 public String load(String key) {
16                     return "Value for " + key; // This method will be used if the key is not present in the cache
17                 }
18             });
19         try {
20             String[] genres = {"Action", "Comedy", "Drama", "Horror", "Romance", "Sci-Fi", "Thriller"};
21
22             // Generate 200,000 movies with random genres and add to cache
23             for (int i = 0; i < 200000; i++) {
24                 String movie = "Movie" + i;
25                 String genre = genres[new Random().nextInt(genres.length)];
26                 cache.put(movie, genre);
27             }
28
29             // Fetch a specific movie multiple times and measure time taken
30             for (int i = 0; i < 100; i++) {
31                 long startTime = System.nanoTime();
32                 String movie = cache.get("Movie92600"); // Change the key if needed
33                 long endTime = System.nanoTime();
34                 System.out.println("Time taken to fetch the movie: " + (endTime - startTime) + " nanoseconds");
35             }
36         } catch (Exception e) {
37             e.printStackTrace();
38         }
39     }
40 }
41 }
```



## Program 27 - Demonstrating Caching Performance Testing

### Product.java

```
P27username > J Product.java > Product > Product(String, String, double)
 1 package P27username;
 2
 3 import java.util.HashMap;
 4 import java.util.Map;
 5 import java.util.Random;
 6
 7 class Product {
 8     private String id;
 9     private String name;
10     private double price;
11
12     public Product(String id, String name, double price) {
13         this.id = id;
14         this.name = name;
15         this.price = price;
16     }
17
18     // Getters and setters omitted for brevity
19
20     @Override
21     public String toString() {
22         return "Product{id='" + id + "', name='" + name + "', price=" + price + "}";
23     }
24 }
25
26 class DatabaseSimulator {
27     private Map<String, Product> products = new HashMap<>();
28     private Random random = new Random();
29
30     public DatabaseSimulator(int numProducts) {
31         for (int i = 0; i < numProducts; i++) {
32             String id = "PROD" + i;
33             products.put(id, new Product(id, "Product " + i, 10 + random.nextDouble() * 90));
34         }
35     }
36
37     public Product getProduct(String id) {
38         // Simulate database access delay
39         try {
40             Thread.sleep(100); // 100ms delay to simulate DB access
41         } catch (InterruptedException e) {
42             e.printStackTrace();
43         }
44         return products.get(id);
45     }
46 }
47
```

## ProductService.java

```
P27username > J ProductService.java > ProductService
1 package P27username;
2
3 import com.google.common.cache.CacheBuilder;
4 import com.google.common.cache.CacheLoader;
5 import com.google.common.cache.LoadingCache;
6
7 import java.util.concurrent.TimeUnit;
8
9 public class ProductService {
10     private final DatabaseSimulator database;
11     private final LoadingCache<String, Product> cache;
12
13     public ProductService(int numProducts, int cacheSize) {
14         this.database = new DatabaseSimulator(numProducts);
15         this.cache = CacheBuilder.newBuilder()
16             .maximumSize(cacheSize)
17             .expireAfterWrite(10, TimeUnit.MINUTES)
18             .recordStats() // This allows us to collect cache statistics
19             .build(new CacheLoader<String, Product>() {
20                 @Override
21                 public Product load(String id) {
22                     return database.getProduct(id);
23                 }
24             });
25     }
26
27     public Product getProduct(String id) throws Exception {
28         return cache.get(id);
29     }
30
31     public void printCacheStats() {
32         System.out.println("Cache stats: " + cache.stats());
33     }
34 }
```

## CachingPerformanceTest.java

```
P27username > J CachingPerformanceTest.java > CachingPerformanceTest
 1 package P27username;
 2
 3 import java.util.Random;
 4
 5 public class CachingPerformanceTest {
 6     Run|Debug
 7     public static void main(String[] args) throws Exception {
 8         testPerformance(numProducts:10_000, cacheSize:1_000); // 10,000 products, 1,000 cache size
 9         testPerformance(numProducts:1_000_000, cacheSize:100_000); // 1 million products, 100,000 cache size
10     }
11
12     private static void testPerformance(int numProducts, int cacheSize) throws Exception {
13         ProductService service = new ProductService(numProducts, cacheSize);
14         Random random = new Random();
15
16         System.out.println("\nTesting with " + numProducts + " products and cache size " + cacheSize);
17
18         // Warm up the cache
19         for (int i = 0; i < 1000; i++) {
20             service.getProduct("PROD" + random.nextInt(numProducts));
21         }
22
23         // Test performance
24         long startTime = System.currentTimeMillis();
25         for (int i = 0; i < 10000; i++) {
26             service.getProduct("PROD" + random.nextInt(numProducts));
27         }
28         long endTime = System.currentTimeMillis();
29
30         System.out.println("Time taken for 10,000 random product retrievals: " + (endTime - startTime) + "ms");
31         service.printCacheStats();
32     }
33 }
```

```
C:\Codebase\Java\Code>javac -cp ".;guava-33.3.0-jre.jar" username/*.java
C:\Codebase\Java\Code>java -cp ".;guava-33.3.0-jre.jar" username.CachingPerformanceTest
Testing with 10000 products and cache size 1000
```

## Program 28 - Demonstrating Hierarchical Caching

### HierarchicalCache.java

```
P27username > P2Busername > J HierarchicalCache.java > ...
1 package P27username.P2Busername;
2 import P27username.Product; // If in the same package, this line is not needed
3 | Ctrl-L to chat, Ctrl-K to generate
4 import com.google.common.cache.Cache;
5 import com.google.common.cache.CacheBuilder;
6 import java.io.*;
7 import java.nio.file.*;
8 import java.util.concurrent.TimeUnit;
9
10 public class HierarchicalCache {
11     private final Cache<String, Product> l1Cache;
12     private final Cache<String, Product> l2Cache;
13     private final Path l3CacheDir;
14
15     public HierarchicalCache(int l1Size, int l2Size, String l3Path) throws IOException {
16         this.l1Cache = CacheBuilder.newBuilder()
17             .maximumSize(l1size)
18             .expireAfterWrite(1, TimeUnit.MINUTES)
19             .recordStats()
20             .build();
21
22         this.l2Cache = CacheBuilder.newBuilder()
23             .maximumSize(l2size)
24             .expireAfterWrite(5, TimeUnit.MINUTES)
25             .recordStats()
26             .build();
27
28         this.l3CacheDir = Paths.get(l3Path);
29         Files.createDirectories(l3CacheDir);
30     }
31
32     public Product get(String key) throws IOException, ClassNotFoundException {
33         // Try L1 Cache
34         Product product = l1Cache.getIfPresent(key);
35         if (product != null) {
36             return product;
37         }
38
39         // Try L2 Cache
40         product = l2Cache.getIfPresent(key);
41         if (product != null) {
42             l1Cache.put(key, product);
43             return product;
44         }
45
46         // Try L3 Cache
47         Path filePath = l3CacheDir.resolve(key);
48         if (Files.exists(filePath)) {
49             try (ObjectInputStream ois = new ObjectInputStream(Files.newInputStream(filePath))) {
50                 product = (Product) ois.readObject();
51                 l2Cache.put(key, product);
52                 l1Cache.put(key, product);
53                 return product;
54             }
55         }
56
57         return null;
58     }
59
60     public void put(String key, Product value) throws IOException {
61         l1Cache.put(key, value);
62         l2Cache.put(key, value);
63
64         // Write to L3 Cache
65         Path filePath = l3CacheDir.resolve(key);
66         try (ObjectOutputStream oos = new ObjectOutputStream(Files.newOutputStream(filePath))) {
67             oos.writeObject(value);
68         }
69     }
70
71     public void printStats() {
72         System.out.println("L1 Cache Stats: " + l1Cache.stats());
73         System.out.println("L2 Cache Stats: " + l2Cache.stats());
74     }
75 }
```

## DataGenerator.java

```
P27username > P28username > J DataGenerator.java > ...
1 package P27username.P28username;
2 import P27username.Product; // If in the same package, this line is not needed
3
4 import java.util.Random;
5
6 public class DataGenerator {
7     private static final Random random = new Random();
8
9     public static Product generateProduct(String id) {
10         return new Product(id, "Product " + id, 10 + random.nextDouble() * 990);
11     }
12
13     public static String generateRandomId(int maxId) {
14         return "PROD" + random.nextInt(maxId);
15     }
16 }
17
```

## HierarchicalCacheTest.java

```
P27username > P28username > J HierarchicalCacheTest.java > ...
1 package P27username.P28username;
2 import P27username.Product; // If in the same package, this line is not needed
3
4
5 import java.io.IOException;
6
7 public class HierarchicalCacheTest {
8     private static final int TOTAL_PRODUCTS = 100_000;
9     private static final int TEST_ITERATIONS = 1_000_000;
10
11     Run|Debug
12     public static void main(String[] args) throws IOException, ClassNotFoundException {
13         HierarchicalCache cache = new HierarchicalCache(l1Size:100, l2Size:1000, l3Path:"l3cache");
14
15         // Populate cache
16         System.out.println("Populating cache...");
17         for (int i = 0; i < TOTAL_PRODUCTS; i++) {
18             String id = "PROD" + i;
19             cache.put(id, DataGenerator.generateProduct(id));
20         }
21
22         // Test random access
23         System.out.println("Testing random access...");
24         long startTime = System.currentTimeMillis();
25
26         for (int i = 0; i < TEST_ITERATIONS; i++) {
27             String randomId = DataGenerator.generateRandomId(TOTAL_PRODUCTS);
28             Product product = cache.get(randomId);
29             if (product == null) {
30                 System.out.println("Product not found: " + randomId);
31             }
32
33             if (i % 100000 == 0) {
34                 System.out.println("Completed " + i + " iterations");
35                 cache.printStats();
36             }
37         }
38
39         long endTime = System.currentTimeMillis();
40         System.out.println("Total time for " + TEST_ITERATIONS + " random accesses: " + (endTime - startTime) + "ms");
41         cache.printStats();
42     }
43 }
```

## Program 29 - Demonstrating Two Level Caching

### TwoLevelCache.java

```
P29 > J TwoLevelCache.java > TwoLevelCache
 1 package P29;
 2 import java.util.HashMap;
 3 import java.util.Map;
 4
 5 public class TwoLevelCache {
 6     private final Map<String, Product> l1Cache;
 7     private final Map<String, Product> l2Cache;
 8     private final int l1Capacity;
 9     private int l1Hits = 0, l2Hits = 0, misses = 0;
10
11     public TwoLevelCache(int l1Capacity, int l2Capacity) {
12         this.l1Cache = new HashMap<>(l1Capacity);
13         this.l2Cache = new HashMap<>(l2Capacity);
14         this.l1Capacity = l1Capacity;
15     }
16
17     public Product get(String key) {
18         // Check L1 Cache
19         if (l1Cache.containsKey(key)) {
20             l1Hits++;
21             return l1Cache.get(key);
22         }
23
24         // Check L2 Cache
25         if (l2Cache.containsKey(key)) {
26             l2Hits++;
27             Product product = l2Cache.get(key);
28             // Move to L1 cache
29             if (l1Cache.size() >= l1Capacity) {
30                 // If L1 is full, remove the first entry (simulating LRU)
31                 String oldestKey = l1Cache.keySet().iterator().next();
32                 l1Cache.remove(oldestKey);
33             }
34             l1Cache.put(key, product);
35             return product;
36         }
37
38         misses++;
39         return null; // Not found in either cache
40     }
41
42     public void put(String key, Product value) {
43         if (l1Cache.size() >= l1Capacity) {
44             // If L1 is full, move the first entry to L2 (simulating LRU)
45             String oldestKey = l1Cache.keySet().iterator().next();
46             l2Cache.put(oldestKey, l1Cache.remove(oldestKey));
47         }
48         l1Cache.put(key, value);
49     }
50
51     public void printStats() {
52         int totalRequests = l1Hits + l2Hits + misses;
53         System.out.println("Cache Stats:");
54         System.out.println("L1 Hits: " + l1Hits + " (" + (l1Hits * 100.0 / totalRequests) + "%)");
55         System.out.println("L2 Hits: " + l2Hits + " (" + (l2Hits * 100.0 / totalRequests) + "%)");
56         System.out.println("Misses: " + misses + " (" + (misses * 100.0 / totalRequests) + "%)");
57     }
58 }
```

## CachingDemo.java

```
P29 > J CachingDemo.java > CachingDemo
 1 package P29;
 2 import java.util.*;
 3 import java.util.concurrent.ConcurrentHashMap;
 4
 5 public class CachingDemo {
 6     // Simulated "database"
 7     private static final Map<Integer, String> database = new HashMap<>();
 8
 9     // Simple cache
10     private static final Map<Integer, String> simpleCache = new HashMap<>();
11
12     // LRU cache
13     private static final int LRU_CAPACITY = 100;
14     private static final LinkedHashMap<Integer, String> lruCache = new LinkedHashMap<Integer, String>(LRU_CAPACITY, 0.75f, true) {
15         protected boolean removeEldestEntry(Map.Entry<Integer, String> eldest) {
16             return size() > LRU_CAPACITY;
17         }
18     };
19
20     // Two-Level cache
21     private static final int L1_CAPACITY = 10;
22     private static final int L2_CAPACITY = 100;
23     private static final Map<Integer, String> l1Cache = new HashMap<>();
24     private static final Map<Integer, String> l2Cache = new HashMap<>();
25
26     // Thread-safe cache
27     private static final Map<Integer, String> concurrentCache = new ConcurrentHashMap<>();
28
29     Run|Debug
30     public static void main(String[] args) {
31         // Populate the "database"
32         for (int i = 0; i < 1000; i++) {
33             database.put(i, "Value" + i);
34         }
35
36         // Test different caching techniques
37         testSimpleCache();
38         testLRUCache();
39         testTwoLevelCache();
40         testConcurrentCache();
41     }
42
43     private static void testSimpleCache() {
44         System.out.println("\nTesting Simple Cache:");
45         long startTime = System.nanoTime();
46         for (int i = 0; i < 1000; i++) {
47             int key = i % 200; // This will cause some cache hits
48             getWithSimpleCache(key);
49         }
50         long endTime = System.nanoTime();
51         System.out.println("Time taken: " + (endTime - startTime) / 1_000_000.0 + " ms");
52         System.out.println("Cache size: " + simpleCache.size());
53     }
54
55     private static void testLRUCache() {
56         System.out.println("\nTesting LRU Cache:");
57         long startTime = System.nanoTime();
58         for (int i = 0; i < 1000; i++) {
59             int key = i % 200; // This will cause some cache hits and evictions
60             getWithLRUCache(key);
61         }
62         long endTime = System.nanoTime();
63         System.out.println("Time taken: " + (endTime - startTime) / 1_000_000.0 + " ms");
64         System.out.println("Cache size: " + lruCache.size());
65     }
66
67     private static void testTwoLevelCache() {
```

```

P29 > J CachingDemo.java > CachingDemo > getWithSimpleCache(int)
  5  public class CachingDemo {
  6      private static void testTwoLevelCache() {
  7          System.out.println("\nTesting Two-Level Cache:");
  8          long startTime = System.nanoTime();
  9          for (int i = 0; i < 10000; i++) {
 10              int key = i % 200; // This will cause hits in both Levels and some misses
 11              getWithTwoLevelCache(key);
 12          }
 13          long endTime = System.nanoTime();
 14          System.out.println("Time taken: " + (endTime - startTime) / 1_000_000.0 + " ms");
 15          System.out.println("L1 Cache size: " + l1Cache.size() + ", L2 Cache size: " + l2Cache.size());
 16      }
 17
 18      private static void testConcurrentCache() {
 19          System.out.println("\nTesting Concurrent Cache:");
 20          long startTime = System.nanoTime();
 21          // Simulate concurrent access with multiple threads
 22          Thread[] threads = new Thread[10];
 23          for (int t = 0; t < threads.length; t++) {
 24              threads[t] = new Thread(() -> {
 25                  for (int i = 0; i < 1000; i++) {
 26                      int key = i % 200;
 27                      getWithConcurrentCache(key);
 28                  }
 29              });
 30              threads[t].start();
 31          }
 32          for (Thread thread : threads) {
 33              try {
 34                  thread.join();
 35              } catch (InterruptedException e) {
 36                  e.printStackTrace();
 37              }
 38          }
 39          long endTime = System.nanoTime();
 40          System.out.println("Time taken: " + (endTime - startTime) / 1_000_000.0 + " ms");
 41          System.out.println("Cache size: " + concurrentCache.size());
 42      }
 43
 44      private static String getWithSimpleCache(int key) {
 45          if (!simpleCache.containsKey(key)) {
 46              simpleCache.put(key, database.get(key));
 47          }
 48          return simpleCache.get(key);
 49      }
 50
 51      private static String getWithLRUCache(int key) {
 52          if (!lruCache.containsKey(key)) {
 53              lruCache.put(key, database.get(key));
 54          }
 55          return lruCache.get(key);
 56      }
 57
 58      private static String getWithTwoLevelCache(int key) {
 59          // Check L1 Cache
 60          if (l1Cache.containsKey(key)) {
 61              return l1Cache.get(key);
 62          }
 63          // Check L2 Cache
 64          if (l2Cache.containsKey(key)) {
 65              String value = l2Cache.get(key);
 66              // Move to L1 cache
 67              if (l1Cache.size() >= L1_CAPACITY) {
 68                  // If L1 is full, remove the first entry (simulating LRU)
 69                  int oldestKey = l1Cache.keySet().iterator().next();
 70                  l1Cache.remove(oldestKey);
 71              }
 72          }
 73      }
 74  }

```

```
131     }
132     l1Cache.put(key, value);
133     return value;
134 }
135 // Not in cache, get from database
136 String value = database.get(key);
137 // Add to L2 cache
138 if (l2Cache.size() >= L2_CAPACITY) {
139     // If L2 is full, remove the first entry (simulating LRU)
140     int oldestKey = l2Cache.keySet().iterator().next();
141     l2Cache.remove(oldestKey);
142 }
143 l2Cache.put(key, value);
144 return value;
```

PROBLEMS 92 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Time taken: 27.7883 ms
Cache size: 200
PS C:\Codebase\Java\Code> ^C
PS C:\Codebase\Java\Code>
PS C:\Codebase\Java\Code> c:; cd 'c:\Codebase\Java\Code'; & 'C:\Program Files\Java\jdk-17.0.1\bin\java' 352c977b15a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P29.CachingDemo'

Testing Simple Cache:
Time taken: 12.5939 ms
Cache size: 200

Testing LRU Cache:
Time taken: 27.2668 ms
Cache size: 100

Testing Two-Level Cache:
Time taken: 166.9995 ms
L1 Cache size: 10, L2 Cache size: 100

Testing Concurrent Cache:
Time taken: 31.5886 ms
Cache size: 200
PS C:\Codebase\Java\Code>
```

# Day 7 - Caching Continue and To Develop a Minor Project

## Program 30 - Demonstrating Caching

```
▶ P30BookLibrayCache.java > P30BookLibrayCache > main(String[])
1 import java.util.concurrent.ConcurrentHashMap;
2 import java.util.concurrent.ConcurrentMap;
3
4
5 import java.util.*;
6
7 public class P30BookLibrayCache {
8     private final Map<String,Book> bookDatabase=new HashMap<>();
9     //Our cache
10    private final Map<String,Book> cache=new HashMap<>();
11
12    private final int CACHE_SIZE=5;
13
14    private int cacheHits=0;
15    private int cacheMisses=0;
16
17    public P30BookLibrayCache(){
18        bookDatabase.put(key:"1",new Book(id:"1",title:"Book1",author:"Author1"));
19        bookDatabase.put(key:"2",new Book(id:"2",title:"Book2",author:"Author2"));
20        bookDatabase.put(key:"3",new Book(id:"3",title:"Book3",author:"Author3"));
21        bookDatabase.put(key:"4",new Book(id:"4",title:"Book4",author:"Author4"));
22        bookDatabase.put(key:"5",new Book(id:"5",title:"Book5",author:"Author5"));
23        bookDatabase.put(key:"6",new Book(id:"6",title:"Book6",author:"Author6"));
24    }
25
26    public Book getBook(String bookId){
27        Book book=cache.get(bookId);
28        if(book!=null){
29            cacheHits++;
30            System.out.println("Cache Hit for bookId: "+bookId);
31            return book;
32        }
33        else{
34            cacheMisses++;
35            System.out.println("Cache Miss for bookId: "+bookId);
36            book=bookDatabase.get(bookId);
37            if(book!=null){
38                cache.put(bookId,book);
39            }
40            return book;
41        }
42    }
43    private void addToCache(String bookId,Book book){
44        if(cache.size()>CACHE_SIZE){
45            String keyToRemove=cache.keySet().iterator().next();
46            cache.remove(keyToRemove);
47            System.out.println("Cache is full. Removing least recently used bookId: "+keyToRemove);
48        }
49        cache.put(bookId,book);
50        System.out.println("Added to cache: "+bookId);
51    }
52    public void printCacheStatistics(){
53        System.out.println("Cache Hits: "+cacheHits);
54        System.out.println("Cache Misses: "+cacheMisses);
55        System.out.println("Cache Hit Ratio: "+((double)cacheHits/(cacheHits+cacheMisses)));
56    }
57}
```

```

55     System.out.println("Cache Hit Ratio: " + (double) cacheHitCount / cacheMissCount);
56     System.out.println("Current Cache Size: " + cache.size());
57     System.out.println("Books in Cache: " + cache.keySet());
58 }
59 private static class Book{
60     private String id;
61     private String title;
62     private String author;
63
64     public Book(String id, String title, String author){
65         this.id=id;
66     }
67     @Override
68     public String toString(){
69         return "Book{id='"+id+"', title='"+title+"', author='"+author+"'}";
70     }
71 }
72 }
73 Run | Debug
74 public static void main(String[] args){
75     P30BookLibraryCache cache=new P30BookLibraryCache();
76     String []requestedBooks={"1","2","3","4","5","6","1","2","3","4","5","6","5","4","3","9"};
77     for(String bookId:requestedBooks){
78
79         Book book=cache.getBook(bookId);
80         if(book!=null){
81             System.out.println("Retrieved book: "+book);
82         } else{
83             System.out.println("Book not found : "+book);
84         }
85         System.out.println("-----");
86     }
87     cache.printCacheStatistics();
88 }
89 }
90 }
91

```

```
PS C:\Codebase\Java\Code> cd ..; cd 'c:\Codebase\Java\Code'; & 'c:\Program Files\Java\jre1.8.0_201\bin\java' -jar 'C:\Codebase\Java\Code\lib\cache.jar'
Cache Miss for bookId: 1
retived book: Book{id='1', title='null', author='null'}
-----
Cache Miss for bookId: 2
retived book: Book{id='2', title='null', author='null'}
-----
Cache Miss for bookId: 3
retived book: Book{id='3', title='null', author='null'}
-----
Cache Miss for bookId: 4
retived book: Book{id='4', title='null', author='null'}
-----
Cache Miss for bookId: 5
retived book: Book{id='5', title='null', author='null'}
-----
Cache Miss for bookId: 6
retived book: Book{id='6', title='null', author='null'}
-----
Cache Hit for bookId: 1
retived book: Book{id='1', title='null', author='null'}
-----
Cache Hit for bookId: 2
retived book: Book{id='2', title='null', author='null'}
-----
Cache Hit for bookId: 3
retived book: Book{id='3', title='null', author='null'}
-----
Cache Hit for bookId: 4
retived book: Book{id='4', title='null', author='null'}
-----
Cache Hit for bookId: 5
retived book: Book{id='5', title='null', author='null'}
-----
Cache Hit for bookId: 6
retived book: Book{id='6', title='null', author='null'}
-----
Cache Hit for bookId: 5
retived book: Book{id='5', title='null', author='null'}
-----
Cache Hit for bookId: 4
retived book: Book{id='4', title='null', author='null'}
-----
Cache Hit for bookId: 3
retived book: Book{id='3', title='null', author='null'}
-----
Cache Miss for bookId: 9
Cache Hit for bookId: 3
retived book: Book{id='3', title='null', author='null'}
-----
Cache Miss for bookId: 9
Book not found : null
Cache Hit for bookId: 3
retived book: Book{id='3', title='null', author='null'}
-----
Cache Miss for bookId: 9
retived book: Book{id='3', title='null', author='null'}
-----
Cache Miss for bookId: 9
Book not found : null
-----
Cache Miss for bookId: 9
Book not found : null
-----
Cache Hits: 9
Cache Misses: 7
Cache Hits: 9
Cache Misses: 7
Cache Hit Ratio: 0.5625
Current Cache Size: 6
Cache Misses: 7
Cache Hit Ratio: 0.5625
Current Cache Size: 6
Cache Hit Ratio: 0.5625
Current Cache Size: 6
Books in Cache: [1, 2, 3, 4, 5, 6]
Current Cache Size: 6
Books in Cache: [1, 2, 3, 4, 5, 6]
Books in Cache: [1, 2, 3, 4, 5, 6]
PS C:\Codebase\Java\Code>
```

## Program 31 - Demonstrating the detailed concepts of caching

```
 1 DetailDocumentCache.java > ...
 2
 3 import java.util.concurrent.ConcurrentHashMap;
 4 import java.util.concurrent.atomic.AtomicInteger;
 5 import java.io.IOException;
 6 import java.nio.file.Files;
 7 import java.nio.file.Path;
 8 import java.nio.file.Paths;
 9 import java.io.Serializable;
10 import java.io.*;
11 import java.util.concurrent.ExecutorService;
12 import java.util.concurrent.Executors;
13 import java.util.concurrent.TimeUnit;
14 //Write comment for each line of code auto generated why its is required
15 public class DetailDocumentCache {
16     //ConcurrentHashMap is used instead of hashmap because it is thread safe and it is used in concurrent environment
17     //It allows multiple threads to access and modify the cache concurrently without causing any issues
18     //Unlike Collections.synchronizedMap, it does not lock the entire map during write operations,
19     //better performance and scalability in multi-threaded environments
20     private final ConcurrentHashMap<String,Document> cache;
21
22     private final String diskStoragePath;
23     private final int maxCacheSize;
24
25     //AtomicInteger is used instead of integer because it provides atomic operations on the integer value
26     //AtomicInteger ensures that the integer value is updated atomically, which means that the value is updated in a single step without any interruptions
27     //This is useful in concurrent environments where multiple threads may be accessing and updating the integer value simultaneously
28     //AtomicInteger provides methods like getAndIncrement, getAndDecrement, compareAndSet, etc., which are thread-safe and do not suffer from the issues of race conditions and inconsistent reads
29     private final AtomicInteger cacheHits=new AtomicInteger(initialValue:0);
30     //Why not volatile instead of AtomicInteger?
31     //Volatile keyword is used to ensure that the value of the variable is always read from the main memory and not from the cache
32     //It ensures that the variable is always up to date and visible to all threads
33     //However, it does not provide atomic operations on the variable
34     //Volatile keyword is useful when you want to ensure that the variable is always read from the main memory and not from the cache
35     //However, it does not provide atomic operations on the variable
36
37     //What is atomic can you give an example?
38     //Atomic operations are operations that are performed in a single step and cannot be interrupted by other threads
39     //AtomicInteger provides methods like getAndIncrement, getAndDecrement, compareAndSet, etc., which are thread-safe and do not suffer from the issues of race conditions and inconsistent reads
40     private final AtomicInteger cacheMisses=new AtomicInteger(initialValue:0);
41
42     public DetailDocumentCache(int maxCacheSize,String diskStoragePath){
43         this.maxCacheSize=maxCacheSize;
44         this.diskStoragePath=diskStoragePath;
45         this.cache=new ConcurrentHashMap<>(maxCacheSize);
46
47         try{
48             Files.createDirectories(Paths.get(diskStoragePath));
49         }catch(IOException e){
50             e.printStackTrace();
51         }
52     }
53
54     public Document getDocument(String documentId) throws IOException,ClassNotFoundException{
55         Document cachedDocument=cache.get(documentId);
56         if(cachedDocument!=null){
```

```

56     |         cacheHits.incrementAndGet();
57     |         return cachedDocument;
58   }
59   else{
60     |         cacheMisses.incrementAndGet();
61     |         Document document=loadDocumentFromDisk(documentId);
62     |         if(document!=null){
63     |             addToCache(documentId,document);
64     |             // evictCacheIfNecessary();
65     |         }
66     |         return document;
67   }
68 }
69
70 }
71 public void saveDocument(Document document) throws IOException{
72   cache.put(document.getDocumentId(),document);
73   evictCacheIfNecessary();
74   saveToDisk(document);
75 }
76 private void saveToDisk(Document document) throws IOException{
77   Path filePath=Paths.get(diskStoragePath,document.getDocumentId());
78   try(ObjectOutputStream out=new ObjectOutputStream(Files.newOutputStream(filePath))){
79     out.writeObject(document);
80   }
81 }
82 public void printCacheStatistics(){
83   System.out.println("Cache Hits: "+cacheHits.get());
84   System.out.println("Cache Misses: "+cacheMisses.get());
85   System.out.println("Cache Size: "+cache.size());
86   System.out.println("Cache Capacity: "+maxCacheSize);
87   System.out.println("Cache Efficiency: "+((double)cacheHits.get()/(cacheHits.get()+cacheMisses.get())*100)+"%");
88   System.out.println("Cache Hit Ratio: "+((double)cacheHits.get()/(cacheHits.get()+cacheMisses.get())*100)+"%");
89   System.out.println("Cache Miss Ratio: "+((double)cacheMisses.get()/(cacheHits.get()+cacheMisses.get())*100)+"%");
90 }
91
92 private void addToCache(String documentId,Document document){
93   if(cache.size()>maxCacheSize){
94     |         evictCacheIfNecessary();
95   }
96   cache.put(documentId,document);
97 }
98 private Document loadDocumentFromDisk(String documentId) throws IOException,ClassNotFoundException{
99   Path filePath=Paths.get(diskStoragePath,documentId);
100  if(Files.exists(filePath)){
101    try(ObjectInputStream in=new ObjectInputStream(Files.newInputStream(filePath))){
102      return (Document)in.readObject();
103    }
104  }
105  return null;
106 }
107 private void evictCacheIfNecessary(){
108   while(cache.size()>maxCacheSize){

```

```

109     String oldestDocument =cache.keySet().iterator().next();
110     cache.remove(oldestDocument);
111 }
112 }
113 //Why static class instead of inner class?
114 //Static class can be instantiated without having to instantiate the outer class
115 //Static class can be used in the outer class without having to instantiate the outer class
116 //Why to implement Serializable interface?
117 //Serializable interface is implemented to allow the Document object to be converted to a byte stream and stored in the file system
118 //This allows the Document object to be saved to the file system and loaded back into the program
119 public static class Document implements Serializable{
120     private final String documentId;
121     private final String content;
122     private final long timestamp;
123     private final long lastModifiedTime;
124
125     public Document(String documentId, String content, long timestamp, long lastModifiedTime){
126         this.documentId=documentId;
127         this.content=content;
128         this.timestamp=timestamp;
129         this.lastModifiedTime=lastModifiedTime;
130     }
131     public String getDocumentId(){
132         return documentId;
133     }
134     public String getContent(){
135         return content;
136     }
137     public long getTimestamp(){
138         return timestamp;
139     }
140     public long getLastModifiedTime(){
141         return lastModifiedTime;
142     }
143     @Override
144     public String toString(){
145         return "Document{"+"documentId='"+documentId+'\''+", content='"+content+'\''+", timestamp='"+timestamp+"', lastModifiedTime='"+lastModifiedTime+''}";
146     }
147 }
148 }
Run|Debug
public static void main(String[] args){
    DetailDocumentCache cache=new DetailDocumentCache(maxCacheSize:100,diskStoragePath:"C:\\Codebase\\my-caching-project\\my-caching-project\\src\\main\\java\\com\\booksystem\\diskStorage");
    //How ExecutorService is better than Thread creating by extending thread class?
    //ExecutorService is better than Thread creating by extending thread class because it provides a more efficient and flexible way to manage threads
    //ExecutorService provides a pool of threads that can be reused to execute multiple tasks
    //It allows tasks to be executed concurrently and provides a way to control the number of threads in the pool
    //It provides a way to schedule tasks to be executed after a delay or at regular intervals
    ExecutorService executor=Executors.newFixedThreadPool(nThreads:2);
    //Simulate multiple threads accessing and modifying the document
    System.out.println("Starting the simulation");
    //Explain these code in detail
    //Simulate multiple threads accessing and modifying the document
}

```

```

149 public static void main(String[] args){
161     for(int i=0;i<100;i++){
162         final int index=i;
163         executor.execute(()->{
164             try{
165                 //Explain these code in detail
166                 //Simulate a document with a unique ID based on the index
167                 //Explain these code in detail
168                 //Simulate a document with a unique ID based on the index
169                 // Explain if and else condition in detail
170                 //If the index is divisible by 10, a new document is created and saved to the cache
171                 //If the index is not divisible by 10, the document is retrieved from the cache
172                 //Explain the purpose of the try and catch block
173                 //The try and catch block is used to catch any exceptions that may be thrown by the code
174                 //The try block is used to execute the code that may throw an exception
175                 //The catch block is used to catch any exceptions that may be thrown by the code
176                 //Explain the purpose of the if and else condition
177                 //The if and else condition is used to check if the index is divisible by 10
178                 //If the index is divisible by 10, a new document is created and saved to the cache
179                 //If the index is not divisible by 10, the document is retrieved from the cache
180             String id = "Document"+ (index);
181             if(index%10==0){
182                 Document newDocument=new Document(id,"Content for "+id,System.currentTimeMillis(),System.currentTimeMillis());
183                 cache.saveDocument(newDocument);
184                 System.out.println("Saved document: "+id);
185             }else{
186                 Document document=cache.getDocument(id);
187                 if(document!=null){
188                     System.out.println("Retrieved document: "+id);
189                 }else{
190                     System.out.println("Document not found: "+id);
191                 }
192             }
193         }catch(IOException | ClassNotFoundException e){
194             e.printStackTrace();
195         }
196     finally{
197         System.out.println("Thread "+index+" completed");
198         // executor.shutdown();
199         try{
200             executor.awaitTermination(timeout:1,TimeUnit.MINUTES);
201         }catch(InterruptedException e){
202             e.printStackTrace();
203         }
204     }
205 }
206 });
207 cache.printCacheStatistics();
208
209 }
210 }
211 }

```



## Program 32 - Book Library System

```
J P32BookLibrarySystem.java > P32BookLibrarySystem > main(String[])
1 import java.util.*;
2 import java.io.*;
3 import java.util.concurrent.*;
4 public class P32BookLibrarySystem {
    Run | Debug
5     public static void main(String[] args) {
6         BookLibrary bookLibrary=new BookLibrary();
7         bookLibrary.addBook(new Book(isbn:"123",title:"Harry Potter",author:"J.K.Rowling"));
8         bookLibrary.addBook(new Book(isbn:"456",title:"To Kill a Mockingbird",author:"Harper Lee"));
9         bookLibrary.addBook(new Book(isbn:"789",title:"1984",author:"George Orwell"));
10        bookLibrary.addBook(new Book(isbn:"101",title:"Pride and Prejudice",author:"Jane Austen"));
11        bookLibrary.addBook(new Book(isbn:"102",title:"The Great Gatsby",author:"F. Scott Fitzgerald"));
12        bookLibrary.addBook(new Book(isbn:"103",title:"Moby Dick",author:"Herman Melville"));
13        bookLibrary.addBook(new Book(isbn:"104",title:"War and Peace",author:"Leo Tolstoy"));
14        bookLibrary.addBook(new Book(isbn:"105",title:"Hamlet",author:"William Shakespeare"));
15        bookLibrary.addBook(new Book(isbn:"106",title:"Macbeth",author:"William Shakespeare"));
16
17        System.out.println(bookLibrary.getAllBooks());
18        System.out.println(bookLibrary.getAuthors());
19        System.out.println(bookLibrary.getBookCountByAuthor());
20        //How Future is used in Asynchronous programming
21        //ExecutorService, Callable and Future
22        //Explain the difference between Future and ExecutorService
23        //Explain the difference between Future and Thread
24        //Explain the difference between Future and Runnable
25
26        Future<Book> futureMostPopularBook=bookLibrary.getMostPopularBookAsync();
27        try{
28            Book mostPopularBook=futureMostPopularBook.get();
29            System.out.println("Most popular book: "+mostPopularBook.toString());
30        }catch(InterruptedException | ExecutionException e){
31            e.printStackTrace();
32        }finally{
33            bookLibrary.executorService.shutdown();
34        }
35        System.out.println("Program continues while waiting for the most popular book");
36    }
37    static class BookLibrary{
38        private List<Book> books = new ArrayList<>();
39
40        private Set<String> authors = new HashSet<>();
41
42        private Map<String,Integer> bookCountByAuthor = new HashMap<>();
43
44        private ExecutorService executorService = Executors.newSingleThreadExecutor();
45        BookLibrary(){
46            books=new ArrayList<>();
47            authors=new HashSet<>();
48            bookCountByAuthor=new HashMap<>();
49            executorService=Executors.newSingleThreadExecutor();
50        }
51        public void addBook(Book book){
52            books.add(book);
53            authors.add(book.author);
54            bookCountByAuthor.put(book.author,bookCountByAuthor.getOrDefault(book.author,defaultValue:0)+1);
55        }
56    }
57}
```

```

37     static class BookLibrary{
38         public void addBook(Book book){
39             authors.add(book.author);
40             bookCountByAuthor.put(book.author,bookCountByAuthor.getOrDefault(book.author,defaultValue:0)+1);
41         }
42         public List<Book> getAllBooks(){
43             return new ArrayList<>(books);
44         }
45         public Set<String> getAuthors(){
46             return new HashSet<>(authors);
47         }
48         public Map<String,Integer> getBookCountByAuthor(){
49             return new HashMap<>(bookCountByAuthor);
50         }
51         //Asynchronous programming
52         public Future<Book> getMostPopularBookAsync(){
53             return executorService.submit(()->{
54                 Thread.sleep(millis:2000);
55                 return books.isEmpty()?null:getMostPopularBook();
56             });
57         }
58         private Book getMostPopularBook(){
59             return books.get(index:0);
60         }
61     }
62     static class Book implements Serializable{
63         private String isbn;
64         private String title;
65         private String author;
66         public Book(String isbn,String title,String author){
67             this.isbn=isbn;
68             this.title=title;
69             this.author=author;
70         }
71         // transient field, it would not be serialized
72         transient int currentPage=0;
73         @Override
74         public String toString(){
75             return "Book{"+"isbn='"+isbn+"', title='"+title+"', author='"+author+"'}";
76         }
77     }
78 }
79 
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```

352-97b15a1b1c113b029ff7a867\redhat.java\jdt_ws\Code_aae1f401\bin\ P32BooklibrarySystem
[Book(isbn='123', title='Harry Potter', author='J.K.Rowling'), Book(isbn='456', title='To Kill a Mockingbird', author='Harper Lee'), Book(isbn='789', title='1984', author='George Orwell'), Book(isbn='101', title='Pride and Prejudice', author='Jane Austen'), Book(isbn='102', title='The Great Gatsby', author='F. Scott Fitzgerald'), Book(isbn='103', title='Moby Dick', author='Herman Melville'), Book(isbn='104', title='War and Peace', author='Leo Tolstoy'), Book(isbn='105', title='Hamlet', author='William Shakespeare'), Book(isbn='106', title='Macbeth', author='William Shakespeare')]
[J.K.Rowling, Harper Lee, Jane Austen, Herman Melville, George Orwell, F. Scott Fitzgerald, Leo Tolstoy, William Shakespeare]
[J.K.Rowling=1, Harper Lee=1, Jane Austen=1, Herman Melville=1, George Orwell=1, F. Scott Fitzgerald=1, Leo Tolstoy=1, William Shakespeare=2]
Most popular book: Book(isbn='123', title='Harry Potter', author='J.K.Rowling')
Program continues while waiting for the most popular book

```

# Day 8 - Data Structures: Array and Sorting

## Program 33 - Detail document of Caching

```
J P33DetailDocumentCache.java > P33DetailDocumentCache > main(String[])
1 import java.util.concurrent.ConcurrentHashMap;
2 import java.util.concurrent.atomic.AtomicInteger;
3 import java.io.IOException;
4 import java.nio.file.Files;
5 import java.nio.file.Path;
6 import java.nio.file.Paths;
7 import java.io.Serializable;
8 import java.io.ObjectInputStream;
9 import java.io.*;
10 import java.util.concurrent.ExecutorService;
11 import java.util.concurrent.Executors;
12 import java.util.concurrent.TimeUnit;
13 import java.util.concurrent.ConcurrentHashMap;
14 import java.util.concurrent.atomic.AtomicInteger;
15 import java.io.IOException;
16 import java.nio.file.Files;
17 import java.nio.file.Path;
18 import java.nio.file.Paths;
19 import java.io.Serializable;
20 import java.io.ObjectInputStream;
21 import java.io.*;
22 import java.util.concurrent.ExecutorService;
23 import java.util.concurrent.Executors;
24 import java.util.concurrent.TimeUnit;
25 //Write comment for each line of code auto generated why its is required
26 public class P33DetailDocumentCache {
27     //ConcurrentHashMap is used instead of hashmap because it is thread safe and it is used in concurrent environment
28     //It allows multiple threads to access and modify the cache concurrently without causing any issues
29     //Unlike Collections.synchronizedMap, it does not lock the entire map during write operations,
30     //better performance and scalability in multi-threaded environments
31     private final ConcurrentHashMap<String,Document> cache;
32
33     private final String diskStoragePath;
34     private final int maxCacheSize;
35
36     //AtomicInteger is used instead of integer because it provides atomic operations on the integer value
37     //AtomicInteger ensures that the integer value is updated atomically, which means that the value is updated in a single step without any interruptions
38     //This is useful in concurrent environments where multiple threads may be accessing and updating the integer value simultaneously
39     //AtomicInteger provides methods like getAndIncrement, getAndDecrement, compareAndSet, etc., which are thread-safe and do not suffer from the issues of race conditions and inconsistent reads
40     private AtomicInteger cacheHits=new AtomicInteger(initialValue:0);
41     //Why not volatile instead of AtomicInteger?
42     //Volatile keyword is used to ensure that the value of the variable is always read from the main memory and not from the cache
43     //It ensures that the variable is always up to date and visible to all threads
44     //However, it does not provide atomic operations on the variable
45     //Volatile keyword is useful when you want to ensure that the variable is always read from the main memory and not from the cache
46     //However, it does not provide atomic operations on the variable
47
48     //What is atomic can you give an example?
49     //Atomic operations are operations that are performed in a single step and cannot be interrupted by other threads
50     //AtomicInteger provides methods like getAndIncrement, getAndDecrement, compareAndSet, etc., which are thread-safe and do not suffer from the issues of race conditions and inconsistent reads
51     private final AtomicInteger cacheMisses=new AtomicInteger(initialValue:0);
52
53     public P33DetailDocumentCache(int maxCacheSize,String diskStoragePath){
54         this.maxCacheSize=maxCacheSize;
55         this.diskStoragePath=diskStoragePath;
56         this.cache=new ConcurrentHashMap<>(maxCacheSize);
57
58         try{
59             Files.createDirectories(Paths.get(diskStoragePath));
60         }catch(IOException e){
61             e.printStackTrace();
62         }
63     }
64
65     public Document getDocumentIfExistElseSave(String documentId,P33DetailDocumentCache cache) throws IOException,ClassNotFoundException{
66         Document cachedDocument=cache.get(documentId);
67         Document newDocument=null;
```

```

68     if(cachedDocument!=null){
69         cacheHits.incrementAndGet();
70         return cachedDocument;
71     }
72     else{
73         cacheMisses.incrementAndGet();
74
75         Document document=loadDocumentFromDisk(documentId);
76         if(document== null){
77             newDocument=new Document(documentId,"Content for "+documentId,System.currentTimeMillis(),System.currentTimeMillis());
78             cache1.saveDocument(newDocument);
79             System.out.println("Saved document: "+documentId);
80             return newDocument;
81         }
82         else{
83             return document;
84         }
85         //System.out.println("getting from disk , id:" + documentId);
86
87     }
88 }
89
90 public void saveDocument(Document document) throws IOException{
91
92     evictCacheIfNecessary();
93     saveToDisk(document);
94     addToCache(document.getDocumentId(),document);
95 }
96
97 private void saveToDisk(Document document) throws IOException{
98     Path filePath=Paths.get(diskStoragePath,document.getDocumentId());
99     try(ObjectOutputStream out=new ObjectOutputStream(Files.newOutputStream(filePath))){
100         out.writeObject(document);
101     }
102 }
103
104 public void printCacheStatistics(){
105     System.out.println("Cache Hits: "+cacheHits.get());
106     System.out.println("Cache Misses: "+cacheMisses.get());
107     System.out.println("Cache Size: "+cache.size());
108     System.out.println("Cache Capacity: "+maxCacheSize);
109     System.out.println("Cache Efficiency: "+((double)cacheHits.get()/(cacheHits.get()+cacheMisses.get())*100)+"%");
110     System.out.println("Cache Hit Ratio: "+((double)cacheHits.get()/(cacheHits.get()+cacheMisses.get())*100)+"%");
111     System.out.println("Cache Miss Ratio: "+((double)cacheMisses.get()/(cacheHits.get()+cacheMisses.get())*100)+"%");
112 }
113
114 private void addToCache(String documentId,Document document){
115     if(cache.size()>maxCacheSize){
116         evictCacheIfNecessary();
117     }
118     cache.put(documentId,document);
119 }
120
121 private Document loadDocumentFromDisk(String documentId) throws IOException,ClassNotFoundException{
122     Path filePath=Paths.get(diskStoragePath,documentId);
123     if(Files.exists(filePath)){
124         try(ObjectInputStream in=new ObjectInputStream(Files.newInputStream(filePath))){
125             return (Document)in.readObject();
126         }
127     }
128     return null;
129 }
130
131 private void evictCacheIfNecessary(){
132     while(cache.size()>maxCacheSize){
133         String oldestDocument =cache.keySet().iterator().next();
134         cache.remove(oldestDocument);
135     }
136 }

```

```
133 }
134 //Why static class instead of inner class?
135 //Static class can be instantiated without having to instantiate the outer class
136 //Static class can be used in the outer class without having to instantiate the outer class
137 //Why to implement Serializable interface?
138 //Serializable interface is implemented to allow the Document object to be converted to a byte stream and stored in the file system
139 //and vice versa. The document is saved to the file system and loaded back into the program
140 public static class Document implements Serializable{
141     private final String documentId;
142     private final String content;
143     private final long timestamp;
144     private final long lastModifiedTime;
145
146     public Document(String documentId, String content, long timestamp, long lastModifiedTime){
147         this.documentId=documentId;
148         this.content=content;
149         this.timestamp=timestamp;
150         this.lastModifiedTime=lastModifiedTime;
151     }
152     public String getDocumentId(){
153         return documentId;
154     }
155     public String getContent(){
156         return content;
157     }
158     public long getTimestamp(){
159         return timestamp;
160     }
161     public long getLastModifiedTime(){
162         return lastModifiedTime;
163     }
164     @Override
165     public String toString(){
166         return "Document{" + "documentId=" + documentId + ", content='" + content + "'\n, timestamp=" + timestamp + ", lastModifiedTime=" + lastModifiedTime + '}';
167     }
168 }
169 }
170 Run|Debug
171 public static void main(String[] args){
172     P3DetailDocumentCache cache=new P3DetailDocumentCache(maxCacheSize:200,diskStoragePath:"C:\\\\Codebase\\\\my-caching-project\\\\my-caching-project\\\\src\\\\main\\\\java\\\\com\\\\cache\\\\bookSystem\\\\diskStorage");
173     //how ExecutorService is better than Thread creating by extending thread class?
174     //ExecutorService is better than Thread creating by extending thread class because it provides a more efficient and flexible way to manage threads
175     //ExecutorService provides a pool of threads that can be reused to execute multiple tasks
176     //It provides a way to control the number of threads in the pool
177     //It provides a way to schedule tasks to be executed after a delay or at regular intervals
178     ExecutorService executor=Executors.newFixedThreadPool(2);
179     //Simulate multiple threads accessing and modifying the document
180     System.out.println("Starting the simulation");
181
182     //Explain these code in detail
183     //Simulate multiple threads accessing and modifying the document
184     for(int i=0;i<20;i++){
185         final int index=(int)i; //Final created to give in Lambda function in executor.
186         executor.execute(()->{
187             try{
188                 //Explain these code in detail
189                 //Simulate a document with a unique ID based on the index
190                 //Explain this code in detail
191                 //Create a document with a unique ID based on the index
192                 //Explain if and else condition in detail
193                 //If the index is divisible by 10, a new document is created and saved to the cache
194                 //If the index is not divisible by 10, the document is retrieved from the cache
195                 //Explain the purpose of the try and catch block
196                 //The try and catch block is used to catch any exceptions that may be thrown by the code
197                 //The try block is used to execute the code that may throw an exception
198                 //The catch block is used to catch any exceptions that may be thrown by the code
199             }catch(Exception e){
200                 e.printStackTrace();
201             }
202         });
203     }
204 }
205
206 //Explain the purpose of the try and catch block
207 //The try and catch block is used to catch any exceptions that may be thrown by the code
208 //The try block is used to execute the code that may throw an exception
209 //The catch block is used to catch any exceptions that may be thrown by the code
210 //Explain the purpose of the if and else condition
211 //The if and else condition is used to check if the index is divisible by 10
212 //If the index is divisible by 10, a new document is created and saved to the cache
213 //If the index is not divisible by 10, the document is retrieved from the cache
214 String id = "Document"+ (Index);
215 if(index%10==0){
216     if(cache.loadDocumentFromDisk(id)!=null){
217         System.out.println("Document already exists: "+id);
218         System.out.println("Document: "+cache.loadDocumentFromDisk(id));
219     }
220     else{
221         Document newDocument=new Document(id,"Content for "+id, System.currentTimeMillis(), System.currentTimeMillis());
222         cache.saveDocument(newDocument);
223         System.out.println("Saved document: "+id);
224     }
225 }else{
226     Document document=cache.getDocumentIfExistElseSave(id,cache);
227     if(document!=null){
228         System.out.println("Retrieved document: "+id);
229     }else{
230         System.out.println("Document not found: "+id);
231     }
232 }
233
234 }catch(IOException | ClassNotFoundException e){
235     e.printStackTrace();
236 }
237 finally{
238     System.out.println("Thread "+index+" completed");
239     // executor.shutdown();
240     try{
241         executor.awaitTermination(timeout:1,TimeUnit.MINUTES);
242         executor.shutdown();
243     }catch(InterruptedException e){
244         e.printStackTrace();
245     }
246 }
247 });
248 cache.printCacheStatistics();
249
250 }
251 }
```

```
194     //Explain the purpose of the try and catch block
195     //The try and catch block is used to catch any exceptions that may be thrown by the code
196     //The try block is used to execute the code that may throw an exception
197     //The catch block is used to catch any exceptions that may be thrown by the code
198     //Explain the purpose of the if and else condition
199     //The if and else condition is used to check if the index is divisible by 10
200     //If the index is divisible by 10, a new document is created and saved to the cache
201     //If the index is not divisible by 10, the document is retrieved from the cache
202     String id = "Document"+ (Index);
203     if(index%10==0){
204         if(cache.loadDocumentFromDisk(id)!=null){
205             System.out.println("Document already exists: "+id);
206             System.out.println("Document: "+cache.loadDocumentFromDisk(id));
207         }
208         else{
209             Document newDocument=new Document(id,"Content for "+id, System.currentTimeMillis(), System.currentTimeMillis());
210             cache.saveDocument(newDocument);
211             System.out.println("Saved document: "+id);
212         }
213     }else{
214         Document document=cache.getDocumentIfExistElseSave(id,cache);
215         if(document!=null){
216             System.out.println("Retrieved document: "+id);
217         }else{
218             System.out.println("Document not found: "+id);
219         }
220     }
221 }catch(IOException | ClassNotFoundException e){
222     e.printStackTrace();
223 }
224 finally{
225     System.out.println("Thread "+index+" completed");
226     // executor.shutdown();
227     try{
228         executor.awaitTermination(timeout:1,TimeUnit.MINUTES);
229         executor.shutdown();
230     }catch(InterruptedException e){
231         e.printStackTrace();
232     }
233 }
234 });
235 cache.printCacheStatistics();
236
237
238
239
240
241 }
```

```

Starting the simulation
Cache Hits: 0
Cache Misses: 0
Cache Size: 0
Cache Capacity: 200
Cache Efficiency: NaN%
Cache Hit Ratio: NaN%
Cache Miss Ratio: NaN%
Cache Hits: 0
Cache Misses: 0
Cache Size: 0
Cache Capacity: 200
Cache Efficiency: 0.0%
Cache Hit Ratio: 0.0%
Cache Miss Ratio: 100.0%
Cache Hits: 0
Cache Misses: 1
Cache Size: 0
Cache Capacity: 200
Cache Efficiency: 0.0%
Cache Hit Ratio: 0.0%
Cache Miss Ratio: 100.0%
Cache Hits: 0
Cache Misses: 1
Cache Size: 0
Cache Capacity: 200
Cache Efficiency: 0.0%
Cache Hit Ratio: 0.0%
Cache Miss Ratio: 100.0%
Cache Hits: 0
Cache Misses: 1
Cache Size: 0
Cache Capacity: 200
Cache Efficiency: 0.0%
Cache Hit Ratio: 0.0%
Cache Miss Ratio: 100.0%
Cache Hits: 0
Cache Misses: 1
Cache Size: 0
Cache Capacity: 200
Cache Efficiency: 0.0%
Cache Hit Ratio: 0.0%
Cache Miss Ratio: 100.0%
Cache Hits: 0
java.lang.ClassNotFoundException: DetailDocumentCache$Document
Cache Misses: 1
    at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:641)
Cache Size: 0
Cache Capacity: 200
Cache Efficiency: 0.0%
    at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:188)
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:520)
    at java.base/java.lang.Class.forName0(Native Method)
    at java.base/java.lang.Class.forName(Class.java:467)
    at java.base/java.io.ObjectInputStream.resolveClass(ObjectInputStream.java:778)
Cache Hit Ratio: 0.0%
Cache Miss Ratio: 100.0%
Cache Hits: 0
Saved document: Document1
Retrieved document: Document1
Cache Misses: 1
Cache Size: 1
Cache Capacity: 200
Cache Efficiency: 0.0%
Thread 1 completed
    at java.base/java.io.ObjectInputStream.readNonProxyDesc(ObjectInputStream.java:2043)
Cache Hit Ratio: 0.0%
    at java.base/java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1907)
    at java.base/java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:2209)
    at java.base/java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1742)
    at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:514)
    at java.base/java.io.ObjectInputStream.readObject0(ObjectInputStream.java:472)
    at P33DetailDocumentCache.loadDocumentFromDisk(P33DetailDocumentCache.java:123)
    at P33DetailDocumentCache.lambda$0(P33DetailDocumentCache.java:284)
    at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1136)
    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635)

```

## Program 34 - LRU Caching Demonstration

```
J P34LFUCacheDemo.java > P34LFUCacheDemo
1 import java.util.*;
2
3 class LFUCache<K, V> {
4     private final int capacity;
5     private Map<K, V> cache;           // For storing cache data
6     private Map<K, Integer> frequencyMap; // For storing frequency of each key
7     private LinkedHashMap<K, Long> accessOrder; // For maintaining LRU (access time)
8
9     public LFUCache(int capacity) {
10         this.capacity = capacity;
11         this.cache = new HashMap<>();
12         this.frequencyMap = new HashMap<>();
13         this.accessOrder = new LinkedHashMap<>(capacity, loadFactor:0.75f, accessOrder:true); // accessOrder = true makes it LRU
14     }
15     public V get(K key) {
16         if (!cache.containsKey(key)) {
17             return null;
18         }
19         // Increment frequency since the item is accessed
20         frequencyMap.put(key, frequencyMap.getOrDefault(key, defaultValue:0) + 1);
21         // Update the access time for LRU handling
22         accessOrder.put(key, System.nanoTime());
23
24         return cache.get(key);
25     }
26
27     public void put(K key, V value) {
28         if (capacity == 0) {
29             return; // No capacity, so no caching
30         }
31
32         if (cache.containsKey(key)) {
33             // If key already exists, just update the value and frequency
34             cache.put(key, value);
35             get(key); // Reuse get method to update frequency and accessOrder
36             return;
37         }
38         // If cache is full, remove the Least frequently used (and LRU if tie)
39         if (cache.size() >= capacity) {
40             evictLFU();
41         }
42         // Add new key-value pair
43         cache.put(key, value);
44         frequencyMap.put(key, value:1); // Initial frequency is 1
45         accessOrder.put(key, System.nanoTime()); // Track access time for LRU
46     }
47     private void evictLFU() {
48         K evictKey = null;
49         int minFrequency = Integer.MAX_VALUE;
50         long oldestAccessTime = Long.MAX_VALUE;
51         // Find the Least frequently used key (and LRU if there's a tie)
52         for (K key : cache.keySet()) {
53             int freq = frequencyMap.getOrDefault(key, defaultValue:0);
54             long accessTime = accessOrder.getOrDefault(key, Long.MAX_VALUE);
55             // Compare frequency first, then fallback on access time for LRU
```

```

55     }
56     // Compare frequency first, then fallback on access time for LRU
57     if (freq < minFrequency || (freq == minFrequency && accessTime < oldestAccessTime)) {
58         minFrequency = freq;
59         oldestAccessTime = accessTime;
60         evictKey = key;
61     }
62     // Remove the selected key from cache, frequency map, and access order
63     if (evictKey != null) {
64         cache.remove(evictKey);
65         frequencyMap.remove(evictKey);
66         accessOrder.remove(evictKey);
67     }
68 }
69 public void displayCache() {
70     System.out.println("Cache: " + cache);
71     System.out.println("Frequency Map: " + frequencyMap);
72     System.out.println("Access Order: " + accessOrder);
73 }
74 }
75
76 public class P34LFUCacheDemo {
77     Run|Debug
78     public static void main(String[] args) {
79         LFUCache<Integer, String> lfuCache = new LFUCache<>(capacity:3);
80
81         lfuCache.put(key:1, value:"A");
82         lfuCache.put(key:2, value:"B");
83         lfuCache.put(key:3, value:"C");
84
85         lfuCache.get(key:1); // Access item 1, frequency increases
86         lfuCache.get(key:1); // Access item 1, frequency increases again
87         lfuCache.get(key:2); // Access item 2, frequency increases
88
89         lfuCache.put(key:4, value:"D"); // This will trigger eviction
90
91         lfuCache.displayCache(); // Display the cache contents
92     }

```

PROBLEMS 109 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:
dhat.java\jdt_ws\Code_aae1f401\bin' 'P34LFUCacheDemo'
Cache: {1=A, 2=B, 4=D}
Frequency Map: {1=3, 2=2, 4=1}
Access Order: {1=612633850835600, 2=612633850839500, 4=612633851153500}
Access Order: {1=612633850835600, 2=612633850839500, 4=612633851153500}
PS C:\Codebase\Java\Code>

```

## DATA STRUCTURES CONCEPTS

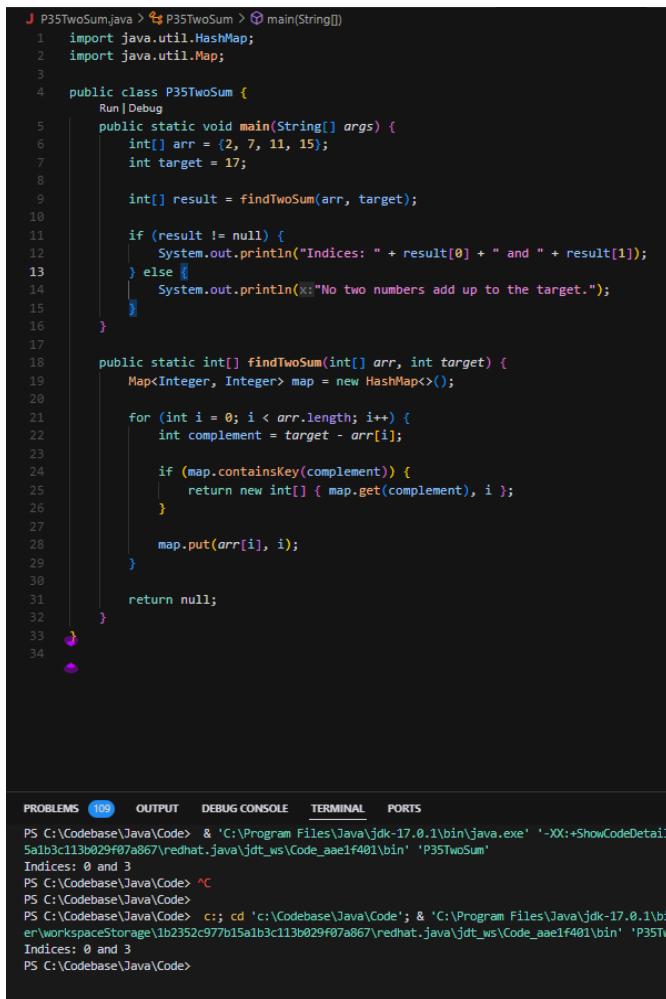
### Array:

An array is a set of uniform data elements that can be accessed using **indexes**.

- An *element* is a single item in an array.
- Uniform means that all elements in a given array must have the same data type.
- The index of an element refers to its position in the array. (Array **indexes** start at 0!)

Arrays can have any number of dimensions. A one-dimensional array is a vector, a two-dimensional array is a table, and a three-dimensional array is a cube. We use the term rank to describe the number of dimensions in an array.

### Program 35 - Demonstration of Two Sum Problem



```
J P35TwoSum.java > P35TwoSum > main(String[])
1 import java.util.HashMap;
2 import java.util.Map;
3
4 public class P35TwoSum {
5     Run|Debug
6     public static void main(String[] args) {
7         int[] arr = {2, 7, 11, 15};
8         int target = 17;
9
10        int[] result = findTwoSum(arr, target);
11
12        if (result != null) {
13            System.out.println("Indices: " + result[0] + " and " + result[1]);
14        } else {
15            System.out.println(":No two numbers add up to the target.");
16        }
17
18    public static int[] findTwoSum(int[] arr, int target) {
19        Map<Integer, Integer> map = new HashMap<>();
20
21        for (int i = 0; i < arr.length; i++) {
22            int complement = target - arr[i];
23
24            if (map.containsKey(complement)) {
25                return new int[] { map.get(complement), i };
26            }
27
28            map.put(arr[i], i);
29        }
30
31        return null;
32    }
33
34 }
```

PROBLEMS 109 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetails'
5a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P35TwoSum'
Indices: 0 and 3
PS C:\Codebase\Java\Code> ^C
PS C:\Codebase\Java\Code>
PS C:\Codebase\Java\Code> c: cd 'c:\Codebase\Java\Code'; & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetails'
5a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P35TwoSum'
Indices: 0 and 3
PS C:\Codebase\Java\Code>
```

## Program 36 - Demonstration of Maximum Subarray using Kadane.

```
J P36MaximumSubarrayKadane.java > ↵ P36MaximumSubarrayKadane > ⚡ maxSubArray(int[])
1  public class P36MaximumSubarrayKadane {
2
3      public static int maxSubArray(int[] arr) {
4          int res = arr[0]; // Initialize result with the first element
5          int maxEnd = arr[0]; // Initialize maxEnd (maximum sum ending at the current index) with the first element
6
7          // Loop through the array starting from the second element
8          for (int i = 1; i < arr.length; i++) {
9              // Update maxEnd, choosing the maximum between the current element or the sum of current element + maxEnd
10             maxEnd = Math.max(maxEnd + arr[i], arr[i]);
11
12             // Update the result (maximum subarray sum so far)
13             res = Math.max(res, maxEnd);
14         }
15
16         return res; // Return the maximum subarray sum
17     }
18
19     Run | Debug
20     public static void main(String[] args) {
21         int[] arr = {2, 3, -8, 7, -1, 2, 3}; // Example input
22         System.out.println("Maximum Subarray Sum: " + maxSubArray(arr)); // Output: 10
23     }
24 }
```

PROBLEMS 109 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> ^C
PS C:\Codebase\Java\Code>
PS C:\Codebase\Java\Code> c;; cd 'c:\Codebase\Java\Code'; & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetailsInException
er\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P36MaximumSubarrayKadane'
Maximum Subarray Sum: 11
PS C:\Codebase\Java\Code> ^C
PS C:\Codebase\Java\Code>
PS C:\Codebase\Java\Code> c;; cd 'c:\Codebase\Java\Code'; & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetailsInException
er\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P36MaximumSubarrayKadane'
Maximum Subarray Sum: 11
PS C:\Codebase\Java\Code>
```

## Program 37 - Demonstration of Container Problem.

```
J P37ContainerWithMostWater.java > P37ContainerWithMostWater
1  public class P37ContainerWithMostWater {
2
3      public static int maxArea(int[] height) {
4          int left = 0, right = height.length - 1;
5          int maxArea = 0;
6
7          while (left < right) {
8              // Calculate the area formed by the Left and right Lines
9              int area = Math.min(height[left], height[right]) * (right - left);
10             // Update the maximum area found
11             maxArea = Math.max(maxArea, area);
12
13             // Move the shorter Line inward
14             if (height[left] < height[right]) {
15                 left++;
16             } else {
17                 right--;
18             }
19         }
20
21         return maxArea;
22     }
23
24     Run | Debug
25     public static void main(String[] args) {
26         int[] height = {1, 8, 6, 2, 5, 4, 8, 3, 7};
27         System.out.println("Maximum water area is: " + maxArea(height)); // Output: 49
28     }
29 }
```

PROBLEMS 109 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionHandler' 'P37ContainerWithMostWater'
Maximum water area is: 49
PS C:\Codebase\Java\Code> ^C
PS C:\Codebase\Java\Code>
PS C:\Codebase\Java\Code> c:; cd 'c:\Codebase\Java\Code'; & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' 'P37ContainerWithMostWater'
Maximum water area is: 49
PS C:\Codebase\Java\Code>
```

## Program 38 - Demonstration of Trapping Rain Water Problem

```
J P38TrappingRainWaterTwoPointer.java > P38TrappingRainWaterTwoPointer > trap(int[])
1  public class P38TrappingRainWaterTwoPointer {
2
3      public static int trap(int[] height) {
4          int left = 0, right = height.length - 1;
5          int leftMax = 0, rightMax = 0;
6          int waterTrapped = 0;
7          Ctrl+L to chat, Ctrl+K to generate
8          // Move pointers from both ends towards the center
9          while (left < right) {
10              if (height[left] < height[right]) {
11                  if (height[left] >= leftMax) {
12                      leftMax = height[left];
13                  } else {
14                      waterTrapped += leftMax - height[left];
15                  }
16                  left++;
17              } else {
18                  if (height[right] >= rightMax) {
19                      rightMax = height[right];
20                  } else {
21                      waterTrapped += rightMax - height[right];
22                  }
23                  right--;
24              }
25          }
26
27          return waterTrapped;
28      }
29
30      Run | Debug
31      public static void main(String[] args) {
32          int[] height = {4, 2, 0, 3, 2, 5};
33          System.out.println("Water trapped: " + trap(height)); // Output: 9
34      }
35 }
```

PROBLEMS 108 OUTPUT DEBUG CONSOLE TERMINAL PORTS  
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetails  
5a1b3c113b029f07a867\redhat.java\jdt\_ws\Code\_aae1f401\bin' 'P38TrappingRainWaterTwoPointer'  
Water trapped: 9  
PS C:\Codebase\Java\Code>

## Program 39 - Demonstration of Recursion

Recursion is a programming technique where a function calls itself in order to solve smaller instances of the same problem. The idea is that a complex problem can often be broken down into simpler sub-problems of the same type.

A recursive function typically has two parts:

1. Base Case: This is the condition that stops the recursion, preventing the function from calling itself indefinitely. Without a base case, the function would run forever, causing a stack overflow error.
2. Recursive Case: This is the part where the function calls itself with a smaller or simpler input, gradually moving towards the base case.

The screenshot shows a Java code editor with the file P39Recursion.java open. The code defines a class P39Recursion with a static method factorial that calculates the factorial of a given number n using recursion. The main method prints the factorial of 5. The terminal below shows the output of the program.

```
J P39Recursion.java > P39Recursion
1 public class P39Recursion {
2     // Recursive method to calculate factorial
3     public static int factorial(int n) {
4         if (n == 0) {
5             return 1; // Base case: factorial of 0 is 1
6         } else {
7             return n * factorial(n - 1); // Recursive case
8         }
9     }
10
11    Run | Debug
12    public static void main(String[] args) {
13        int num = 5;
14        int result = factorial(num);
15        System.out.println("Factorial of " + num + " is: " + result);
16    }
17
18 }
```

PROBLEMS 109 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-oamino\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.java'
Factorial of 5 is: 120
PS C:\Codebase\Java\Code>
```

## Program 40 - Demonstration of Selection Sorting

Selection Sort is a simple comparison-based sorting algorithm. It works by repeatedly finding the smallest (or largest) element from the unsorted part of the array and swapping it with the first element of the unsorted portion. This process continues, gradually building a sorted section at the beginning of the array, while reducing the unsorted part. Although easy to understand and implement, Selection Sort is inefficient for large datasets due to its time complexity of  $O(n^2)$ , making it more suitable for small or nearly sorted arrays.

```
J P40SelectionSort.java > P40SelectionSort
1  public class P40SelectionSort {
2      // Method to perform selection sort
3      public static void selectionSort(int[] arr) {
4          int n = arr.length;
5
6          // Go through each element of the array
7          for (int i = 0; i < n - 1; i++) {
8              // Assume the current element is the smallest
9              int minIndex = i;
10
11             // Find the smallest element in the rest of the array
12             for (int j = i + 1; j < n; j++) {
13                 if (arr[j] < arr[minIndex]) {
14                     minIndex = j; // Update minIndex if a smaller element is found
15                 }
16             }
17
18             // Swap the smallest element found with the current element
19             int temp = arr[minIndex];
20             arr[minIndex] = arr[i];
21             arr[i] = temp;
22         }
23     }
24
25     // Method to print the array
26     public static void printArray(int[] arr) {
27         for (int num : arr) {
28             System.out.print(num + " ");
29         }
30         System.out.println();
31     }
32
33     Run | Debug
34     public static void main(String[] args) {
35         int[] arr = {29, 10, 14, 37, 13};
36
37         System.out.println("Before Sorting:");
38         printArray(arr);
39
40         // Call selectionSort method
41         selectionSort(arr);
42
43         System.out.println("After Sorting:");
44         printArray(arr);
45     }
46 }
```

PROBLEMS 109 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
5ab13c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P40SelectionSort'
Before Sorting:
29 10 14 37 13
After Sorting:
10 13 14 29 37
PS C:\Codebase\Java\Code>
```

## Program 41 - Demonstration of Bubble Sorting

Bubble Sort is a simple sorting algorithm that works by repeatedly swapping adjacent elements if they are in the wrong order. Starting from the beginning of the array, it compares each pair of adjacent elements and "bubbles" the larger (or smaller) element towards the end. This process is repeated for each element in the array until the entire array is sorted. Although easy to implement, Bubble Sort is inefficient for large datasets due to its average and worst-case time complexity of  $O(n^2)$ , making it suitable mainly for small or nearly sorted arrays.

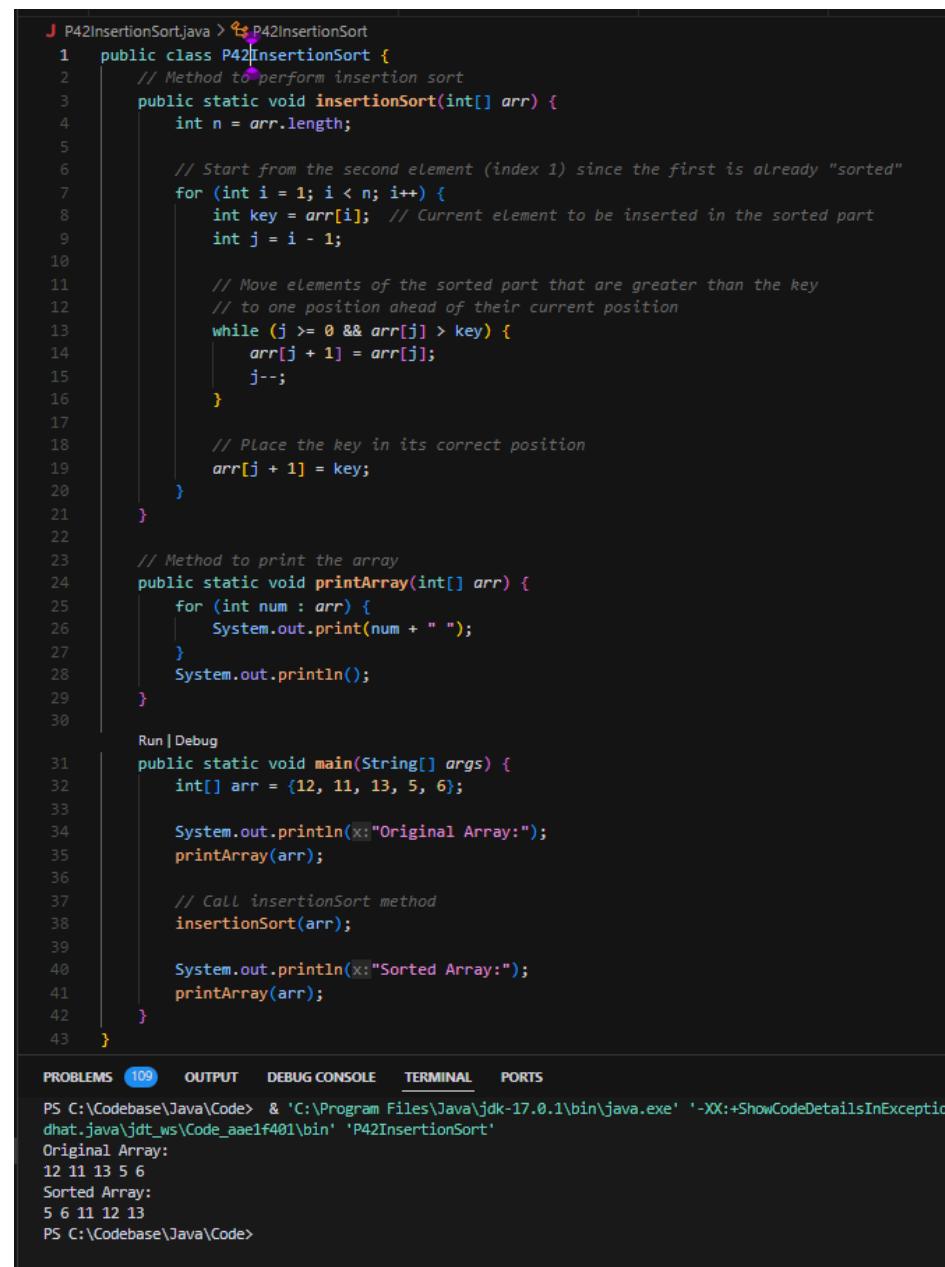
```
J P41BubbleSort.java > P41BubbleSort
1 public class P41BubbleSort {
2     // Method to perform bubble sort
3     public static void bubbleSort(int[] arr) {
4         int n = arr.length;
5
6         // Outer loop to control the number of passes
7         for (int i = 0; i < n - 1; i++) {
8             // Inner loop to perform comparisons and swaps
9             for (int j = 0; j < n - i - 1; j++) {
10                 // Swap if the current element is greater than the next element
11                 if (arr[j] > arr[j + 1]) {
12                     int temp = arr[j];
13                     arr[j] = arr[j + 1];
14                     arr[j + 1] = temp;
15                 }
16             }
17         }
18     }
19
20     // Method to print the array
21     public static void printArray(int[] arr) {
22         for (int num : arr) {
23             System.out.print(num + " ");
24         }
25         System.out.println();
26     }
27
28     Run | Debug
29     public static void main(String[] args) {
30         int[] arr = {64, 34, 25, 12, 22, 11, 90};
31
32         System.out.println("Original Array:");
33         printArray(arr);
34
35         // Call bubbleSort method
36         bubbleSort(arr);
37
38         System.out.println("Sorted Array:");
39         printArray(arr);
40     }
41 }
```

PROBLEMS 109 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Original Array:  
64 34 25 12 22 11 90  
Sorted Array:  
11 12 22 25 34 64 90  
PS C:\Codebase\Java\Code>

## Program 42 - Demonstration of Insertion Sorting

Insertion Sort is a simple and efficient algorithm that works by building a sorted array one element at a time. It starts by assuming that the first element is already sorted, then takes the next element and inserts it into its correct position within the sorted part of the array. This process continues until all elements are sorted. While its average and worst-case time complexity is **O(n<sup>2</sup>)**, Insertion Sort performs well for small datasets or nearly sorted arrays and is often used in practice for these cases due to its simplicity and efficiency on partially sorted data.



```
J P42InsertionSort.java > P42InsertionSort
1  public class P42InsertionSort {
2      // Method to perform insertion sort
3      public static void insertionSort(int[] arr) {
4          int n = arr.length;
5
6          // Start from the second element (index 1) since the first is already "sorted"
7          for (int i = 1; i < n; i++) {
8              int key = arr[i]; // Current element to be inserted in the sorted part
9              int j = i - 1;
10
11             // Move elements of the sorted part that are greater than the key
12             // to one position ahead of their current position
13             while (j >= 0 && arr[j] > key) {
14                 arr[j + 1] = arr[j];
15                 j--;
16             }
17
18             // Place the key in its correct position
19             arr[j + 1] = key;
20         }
21     }
22
23     // Method to print the array
24     public static void printArray(int[] arr) {
25         for (int num : arr) {
26             System.out.print(num + " ");
27         }
28         System.out.println();
29     }
30
31     Run | Debug
32     public static void main(String[] args) {
33         int[] arr = {12, 11, 13, 5, 6};
34
35         System.out.println("Original Array:");
36         printArray(arr);
37
38         // Call insertionSort method
39         insertionSort(arr);
40
41         System.out.println("Sorted Array:");
42         printArray(arr);
43     }
44 }
45
46 PROBLEMS 109 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionDialogs' 'P42InsertionSort'
Original Array:
12 11 13 5 6
Sorted Array:
5 6 11 12 13
PS C:\Codebase\Java\Code>
```

## Day 9 - Arrays, SQL Basics and Database Connectivity

### Program 43 - Remove Duplicates from an Array

```
J P43RemoveDuplicates.java > P43RemoveDuplicates
1 import java.util.*;
2 public class P43RemoveDuplicates {
3     Run | Debug
4     public static void main(String[] args) {
5         int arr[] = {1,1,2,2,2,3,3};
6         int k = removeDuplicates(arr);
7         System.out.println("The array after removing duplicate elements is ");
8         for (int i = 0; i < k; i++) {
9             System.out.print(arr[i] + " ");
10        }
11    }
12    static int removeDuplicates(int[] arr) {
13        HashSet < Integer > set = new HashSet < > ();
14        for (int i = 0; i < arr.length; i++) {
15            set.add(arr[i]);
16        }
17        int k = set.size();
18        int j = 0;
19        for (int x: set) {
20            arr[j++] = x;
21        }
22    }
23 }
```

PROBLEMS 109 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDeta
sages' '-cp' 'C:\Users\Anjali\AppData\Roaming\Cursor\User\workspaceStorage\1b2352c977b15a1b3c11
at.java\jdt_ws\Code_aae1f401\bin' 'P43RemoveDuplicates'
The array after removing duplicate elements is
1 2 3
PS C:\Codebase\Java\Code>
```

## Program 44 - Rotate an Array

```
J P44RotateArray.java > P44RotateArray > main(String[])
1 import java.util.Arrays;
2 public class P44RotateArray {
3
4     // Function to rotate the array
5     public static void rotate(int[] arr, int steps) {
6         // Get the length of the array
7         int n = arr.length;
8
9         // Normalize the number of steps (in case it's larger than array size)
10        steps = steps % n;
11
12        // Reverse the whole array
13        reverse(arr, start:0, n - 1);
14
15        // Reverse the first part (0 to steps-1)
16        reverse(arr, start:0, steps - 1);
17
18        // Reverse the second part (steps to n-1)
19        reverse(arr, steps, n - 1);
20    }
21
22    // Helper function to reverse a portion of the array
23    private static void reverse(int[] arr, int start, int end) {
24        while (start < end) {
25            int temp = arr[start];
26            arr[start] = arr[end];
27            arr[end] = temp;
28            start++;
29            end--;
30        }
31    }
32    // Main function to test the rotation
33    public static void main(String[] args) {
34        int[] arr = {1, 2, 3, 4, 5, 6, 7};
35        int steps = 3; // Number of positions to rotate
36
37        System.out.println("Original Array: " + Arrays.toString(arr));
38
39        rotate(arr, steps);
40
41        System.out.println("Rotated Array: " + Arrays.toString(arr));
42    }
43 }
```

PROBLEMS 109 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetailsInError' '5a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P44RotateArray'
Original Array: [1, 2, 3, 4, 5, 6, 7]
Rotated Array: [5, 6, 7, 1, 2, 3, 4]
PS C:\Codebase\Java\Code>
```

## Program 45 - Moving all zeros to end of an Array

```
J P45ZeroEnd.java > P45ZeroEnd > main(String[])
1  public class P45ZeroEnd {
2      Run| Debug
3      public static void main(String[] args) {
4          int[] arr = {1, 2, 0, 0, 3, 4, 0};
5          int zeroCount = 0;
6
7          for (int i = 0; i < arr.length; i++) {
8              if (arr[i] == 0) {
9                  zeroCount++;
10             }
11         }
12
13
14         int[] newArray = new int[arr.length];
15         int index = 0;
16
17
18         for (int i = 0; i < arr.length; i++) {
19             if (arr[i] != 0) {
20                 newArray[index++] = arr[i];
21             }
22         }
23
24
25         for (int i = index; i < newArray.length; i++) {
26             newArray[i] = 0;
27         }
28
29
30         System.out.print("Output: ");
31         for (int i = 0; i < newArray.length; i++) {
32             System.out.print(newArray[i]);
33         }
34     }
35 }
```

PROBLEMS 110 OUTPUT DEBUG CONSOLE TERMINAL PORTS

aming\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redha
Output: 1234000
PS C:\Codebase\Java\Code>

## Spring Boot

1. Stand Alone
2. Production-grade

Spring Based Application

- Web
- API

## Program 46 - Hash Set of an Array

```
J P46UniqueElement.java > P46UniqueElement > main(String[])
1 import java.util.*;
2 public class P46UniqueElement {
3     Run|Debug
4     public static void main(String[] args) {
5         int[] arr = {1,1,2,2,3,3,3,4,4,4};
6         HashSet <Integer> set = new HashSet<>();
7         for (int i = 0; i < arr.length; i++) {
8             set.add(arr[i]);
9         }
10        System.out.println(set);
11        System.out.println(set.size());
12    }
}
```

PROBLEMS 110 OUTPUT DEBUG CONSOLE TERMINAL PORTS  
oaming\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\re  
[1, 2, 3, 4]  
4  
PS C:\Codebase\Java\Code>

## Installing MySQL Workbench

- <https://dev.mysql.com/downloads/installer/>

Creating Database in MySQL Workbench:

The screenshot shows the MySQL Workbench interface with the following details:

**Query Editor (Query 1):**

```
1 •  create database employeemanagement;
2 •  show databases;
3 •  use employeemanagement;
4 •  create table employees(
5     id int auto_increment primary key,
6     first_name varchar(50),
7     last_name varchar(50),
8     email varchar(100),
9     hire_date DATE,
10    salary decimal(10,2)
11 );
12 •  select * from employees;
13 •  insert into employees (first_name, last_name, email, hire_date, salary)
```

**Result Grid:**

ID	First Name	Last Name	Email	Hire Date	Salary
1	John	Doe	john@gmail.com	2023-05-15	73774.34
2	A	Doe	A@gmail.com	2023-05-15	73774.34
*	HULL	HULL	HULL	HULL	HULL

**Action Output:**

#	Time	Action	Message
5	13:40:00	show databases	27 row(s) returned
6	13:40:09	use employeemanagement	0 row(s) affected
7	13:40:14	create table employees(id int auto_increment primary key, first_name varchar(50), last_name varchar(50), email varchar(100), hire_date DATE, salary decimal(10,2))	0 row(s) affected
8	13:40:20	select * from employees LIMIT 0, 1000	0 row(s) returned
9	13:40:28	insert into employees (first_name, last_name, email, hire_date, salary) values ('John', 'Doe', 'john@gmail.com', '2023-05-15', 73774.34)	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
10	13:40:39	select * from employees LIMIT 0, 1000	2 row(s) returned

## Program 47 - MySQL CRUD Operations

```
J P47CRUDMySQL.java > ...
1  import java.sql.*;
2  public class P47CRUDMySQL {
3      private static final String URL = "jdbc:mysql://localhost:3306/employeemanagement";
4      private static final String USERNAME = "root";
5      private static final String PASSWORD = "root";
6      //Generate comment for all the instance variables
7
8      private static Connection connection;
9      private static Statement statement;
10     private static ResultSet resultSet;
11     private static PreparedStatement preparedStatement;
12
13     Run | Debug
14     public static void main(String[] args) {
15         try {
16             connection = DriverManager.getConnection(URL, USERNAME, PASSWORD);
17
18             createRecord();
19             readRecord();
20             updateRecord();
21             deleteRecord();
22         } catch (SQLException e) {
23             e.printStackTrace();
24         }finally{
25             try {
26                 if (resultSet != null) {
27                     resultSet.close();
28                 }
29                 if (statement != null) {
30                     statement.close();
31                 }
32                 if (connection != null) {
33                     connection.close();
34                 }
35                 if(preparedStatement != null){
36                     preparedStatement.close();
37                 }
38             } catch (SQLException e) {
39                 e.printStackTrace();
40             }
41         }
42     }
43     //following is my create table statement
44     /*create table employees(
45     id int auto_increment primary key,
46     first_name varchar(50),
47     last_name varchar(50),
48     email varchar(100),
49     hire_date DATE,
50     salary decimal(10,2)
51 ); */
52     private static void createRecord(){
53         String query = "INSERT INTO employees (first_name, last_name, email, hire_date, salary) VALUES (?, ?, ?, ?, ?)";
54         try {
```

```

54     try {
55         preparedStatement = connection.prepareStatement(query);
56         preparedStatement.setString(parameterIndex:1, x:"Jinesh");
57         preparedStatement.setString(parameterIndex:2, x:"Doe");
58         preparedStatement.setString(parameterIndex:3, x:"john.doe@example.com");
59         preparedStatement.setDate(parameterIndex:4, new Date(System.currentTimeMillis()));
60         preparedStatement.setDouble(parameterIndex:5, x:50000.00);
61         preparedStatement.executeUpdate();
62         System.out.println(x:"Record created successfully.");
63     } catch (SQLException e) {
64         e.printStackTrace();
65     }
66 }
67 private static void readRecord(){
68     String query = "SELECT * FROM employees";
69     try {
70         statement = connection.createStatement();
71         resultSet = statement.executeQuery(query);
72         while (resultSet.next()) {
73             System.out.println(resultSet.getString(columnLabel:"first_name") + " " + resultSet.getString(columnLabel:"last_name"));
74         }
75     } catch (SQLException e) {
76         e.printStackTrace();
77     }
78 }
79 private static void updateRecord(){
80     String query = "UPDATE employees SET salary = ? WHERE id = ?";
81     try {
82         preparedStatement = connection.prepareStatement(query);
83         preparedStatement.setDouble(parameterIndex:1, x:60000.00);
84         preparedStatement.setInt(parameterIndex:2, x:4);
85         preparedStatement.executeUpdate();
86         System.out.println(x:"Record updated successfully.");
87     } catch (SQLException e) {
88         e.printStackTrace();
89     }
90 }
91 private static void deleteRecord(){
92     String query = "DELETE FROM employees WHERE id = ?";
93     try {
94         preparedStatement = connection.prepareStatement(query);
95         preparedStatement.setInt(parameterIndex:1, x:1);
96         preparedStatement.executeUpdate();
97         System.out.println(x:"Record deleted successfully.");
98     } catch (SQLException e) {
99         e.printStackTrace();
100    }
101 }
102 }
```

# Day 10 - Spring Concepts and Group Project

## Program 48 - Two Sum

```
J TwoSum.java > ...
1 import java.util.HashMap;
2
3 public class TwoSum {
4     public static int[] twoSum(int[] nums, int target) {
5         HashMap<Integer, Integer> map = new HashMap<>();
6         for (int i = 0; i < nums.length; i++) {
7             int complement = target - nums[i];
8             if (map.containsKey(complement)) {
9                 return new int[]{map.get(complement), i};
10            }
11            map.put(nums[i], i);
12        }
13        throw new IllegalArgumentException("No two sum solution");
14    }
15
16    Run | Debug
17    public static void main(String[] args) {
18        int[] nums = {2, 7, 11, 15};
19        int target = 9;
20        int[] result = twoSum(nums, target);
21        System.out.println("Indices: " + result[0] + ", " + result[1]);
22    }
23 }
```

PROBLEMS 110 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> c:; cd 'c:\Codebase\Java\Code'; & 'C:\Program Files\Java\jdk-22\bin\java.exe' 'C:\Users\Anjali\AppData\Roaming\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.
Indices: 0, 1
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInException\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin'
Indices: 0, 1
PS C:\Codebase\Java\Code>
```

## Program 49 - Array Sorting Program

```
J P49ArraySorting.java > P49ArraySorting
1 import java.util.Arrays;
2
3 public class P49ArraySorting {
4     Run | Debug
5     public static void main(String[] args) {
6         int[] array = {5, 2, 8, 1, 3};
7         Arrays.sort(array);
8         System.out.println("Sorted Array: " + Arrays.toString(array));
9     }
10 }
```

PROBLEMS 110 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+S
ng\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.java\jdt
Sorted Array: [1, 2, 3, 5, 8]
PS C:\Codebase\Java\Code>
```

---

## Spring Concepts:

**Spring MVC** (formally, **Spring Web MVC**) is a framework for creating web applications. It allows us to write simple Java code that generates HTTP responses from requests.

It implements the MVC pattern:

- A well-formed HTTP request activates a controller method. A controller is a plain old Java class with **methods** that handle requests.
- The controller method executes actions that result in data. That data is the model.
- The model is merged into a view. The result is included in the HTTP response as content.

**Spring MVC** makes heavy use of Spring dependency injection. It can be configured via XML or Java annotations.

MVC —> Web Application

Model - Brain of your code/application

- Data & Business Logic

View - User Sees

Controller - Model

## Aspect Oriented Programming (AOP):

Aspect-Oriented Programming (AOP) is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. These are aspects of a program that affect multiple modules but are not part of the primary business logic, such as logging, security, and error handling.

@aspect

Class

{

  @Before(  )

  @After(  )

  Public Static helloworld()

{

```
Syop( )
```

```
Return "
```

```
}
```

```
}
```

## Spring - Open Source Framework

Core Feature : DI - Dependency Injection

### Program 50 - Demonstrating Dependency Injection

```
J P50DI.java > P50DI
1  public class P50DI {
2      interface Chef {
3          void prepareDish();
4      }
5
6      static class ItalianChef implements Chef {
7          @Override
8          public void prepareDish() {
9              System.out.println("Preparing Italian dish!");
10         }
11     }
12
13     static class FrenchChef implements Chef {
14         @Override
15         public void prepareDish() {
16             System.out.println("Preparing French dish!");
17         }
18     }
19
20     static class Restaurant {
21         private Chef chef;
22
23         public Restaurant(Chef chef) {
24             this.chef = chef;
25         }
26
27         public void serveDish() {
28             System.out.println("Welcome to the restaurant!");
29             chef.prepareDish();
30         }
31     }
32
33     Run | Debug
34     public static void main(String[] args) {
35         Chef italianChef = new ItalianChef();
36         Restaurant italianRestaurant = new Restaurant(italianChef);
37         italianRestaurant.serveDish();
38
39         Chef frenchChef = new FrenchChef();
40         Restaurant frenchRestaurant = new Restaurant(frenchChef);
41         frenchRestaurant.serveDish();
42     }
43 }
```

PROBLEMS 110 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Welcome to the restaurant!
Preparing Italian dish!
Welcome to the restaurant!
Preparing French dish!
PS C:\Codebase\Java\Code>
```

## Program 51 - Demonstrating Inversion Controller

```
P51InversionControl.java > P51InversionControl
1  public class P51InversionControl {
2
3      interface Service {
4          void execute();
5      }
6
7      static class EmailService implements Service {
8          @Override
9          public void execute() {
10              System.out.println("Sending email...");
11         }
12     }
13
14     static class SmsService implements Service {
15         @Override
16         public void execute() {
17             System.out.println("Sending SMS...");
18         }
19     }
20
21     static class Notification {
22         private Service service;
23
24         public Notification(Service service) {
25             this.service = service; // Dependency injection
26         }
27
28         public void notifyUser() {
29             service.execute();
30         }
31     }
32
33     public static void main(String[] args) {
34         Service emailService = new EmailService();
35         Notification notification1 = new Notification(emailService);
36         notification1.notifyUser();
37
38         Service smsService = new SmsService();
39         Notification notification2 = new Notification(smsService);
40         notification2.notifyUser();
41     }
42 }
```

PROBLEMS 110    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInError' 'P51InversionControl'
3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P51InversionControl'
Sending email...
Sending SMS...
PS C:\Codebase\Java\Code>
```

## Spring Boot Application

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
public class P52Spring {

    public static void main(String[] args) {
        P52Spring.run(P52Spring.class, args);
    }

    @RestController
    static class HelloController {
        @GetMapping("/hello")
        public String hello() {
            return "Hello, Spring Boot!";
        }
    }
}
```

## Program 52 - Demonstrating Stock buying and selling returning maximum profit

```
J P52Stock.java > P52Stock
1  public class P52Stock {
2      Run|Debug
3      public static void main(String[] args) {
4          int[] prices = {7, 1, 5, 3, 6, 4}; // Sample prices
5          System.out.println("Maximum Profit: " + maxProfit(prices));
6      }
7
8      public static int maxProfit(int[] prices) {
9          if (prices.length == 0) return 0;
10
11         int maxProfit = 0;
12         int minPrice = prices[0];
13
14         for (int price : prices) {
15             if (price < minPrice) {
16                 minPrice = price; // Update minimum price
17             } else {
18                 maxProfit = Math.max(maxProfit, price - minPrice); // Calculate profit
19             }
20         }
21     }
22 }
23 }
24 }
```

PROBLEMS 110 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionSpaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P52Stock'
Maximum Profit: 5
PS C:\Codebase\Java\Code>
```

## Program 53 - Demonstrating Longest Consecutive Subsequence

```
J P53LongestConsecutive.java > P53LongestConsecutive
1 import java.util.HashSet;
2
3 public class P53LongestConsecutive {
4     Run | Debug
5     public static void main(String[] args) {
6         int[] nums = {100, 4, 200, 1, 3, 2}; // Sample input
7         System.out.println("Length of Longest Consecutive Subsequence: " + longestConsecutive(nums));
8     }
9
10    public static int longestConsecutive(int[] nums) {
11        HashSet<Integer> numSet = new HashSet<>();
12        for (int num : nums) {
13            numSet.add(num);
14        }
15
16        int longestStreak = 0;
17
18        for (int num : nums) {
19            if (!numSet.contains(num - 1)) { // Check if it's the start of a sequence
20                int currentNum = num;
21                int currentStreak = 1;
22
23                while (numSet.contains(currentNum + 1)) { // Count consecutive numbers
24                    currentNum++;
25                    currentStreak++;
26                }
27
28                longestStreak = Math.max(longestStreak, currentStreak); // Update longest streak
29            }
30        }
31
32        return longestStreak;
33    }
34}
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase\Java\Code> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages'
g\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P53LongestCo
Length of Longest Consecutive Subsequence: 4
PS C:\Codebase\Java\Code>
```

## Program 54 - Print all the subarrays whose sum is 6.

```
J P54Subaaray.java > ↵ P54Subaaray
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class P54Subaaray {
    Run|Debug
5     public static void main(String[] args) {
6         int[] nums = {1, 2, 3, 4, 5}; // Sample input
7         int targetSum = 6;
8         List<int[]> result = findSubarraysWithSum(nums, targetSum);
9
10        System.out.println("Subarrays with sum " + targetSum + ":");
11        for (int[] subarray : result) {
12            System.out.print(s:"[");
13            for (int num : subarray) {
14                System.out.print(num + " ");
15            }
16            System.out.println(x:")");
17        }
18    }
19
20    public static List<int[]> findSubarraysWithSum(int[] nums, int targetSum) {
21        List<int[]> subarrays = new ArrayList<>();
22
23        for (int start = 0; start < nums.length; start++) {
24            int currentSum = 0;
25            for (int end = start; end < nums.length; end++) {
26                currentSum += nums[end];
27                if (currentSum == targetSum) {
28                    int[] subarray = new int[end - start + 1];
29                    System.arraycopy(nums, start, subarray, destPos:0, subarray.length);
30                    subarrays.add(subarray);
31                }
32            }
33        }
34
35        return subarrays;
36    }
37 }
```

PROBLEMS 110 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
g\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin'
Subarrays with sum 6:
Subarrays with sum 6:
[1 2 3 ]
```

## Program 55 - Demonstrating Prefix Sum

```
J P55PrefixSum.java > P55PrefixSum
1 import java.util.Arrays;
2
3 public class P55PrefixSum {
4     Run|Debug
5     public static void main(String[] args) {
6         int[] nums = {1, 2, 3, 4, 5}; // Sample input
7         int[] prefixSum = calculatePrefixSum(nums);
8
9         System.out.println("Original Array: " + Arrays.toString(nums));
10        System.out.println("Prefix Sum Array: " + Arrays.toString(prefixSum));
11
12        // Example: Calculate sum of subarray from index 1 to 3 (inclusive)
13        int start = 1, end = 3; // subarray: [2, 3, 4]
14        int sum = getSubarraySum(prefixSum, start, end);
15        System.out.println("Sum of subarray from index " + start + " to " + end + ": " + sum);
16    }
17
18    public static int[] calculatePrefixSum(int[] nums) {
19        int n = nums.length;
20        int[] prefixSum = new int[n + 1]; // Prefix sum array of size n + 1
21
22        for (int i = 0; i < n; i++) {
23            prefixSum[i + 1] = prefixSum[i] + nums[i]; // Calculate prefix sum
24        }
25
26        return prefixSum;
27    }
28
29    public static int getSubarraySum(int[] prefixSum, int start, int end) {
30        return prefixSum[end + 1] - prefixSum[start]; // Calculate subarray sum
31    }
32 }
```

PROBLEMS 110 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
ng\Cursor\User\workspaceStorage\1b2352c977b15a1b3c113b029f07a867\redhat.java\jdt_ws\Code_aae1f401\bin' 'P55P
Original Array: [1, 2, 3, 4, 5]
Prefix Sum Array: [0, 1, 3, 6, 10, 15]
Sum of subarray from index 1 to 3: 9
PS C:\Codebase\Java\Code>
```

---

## **Group activity**

### **E-commerce Application Documentation**

<https://docs.google.com/document/d/1WDNj2o7EvFAmYtei3WzCc-242LkAteDY2KJrdeBQ2bQ/edit?usp=sharing>

# Day 11 - Maven Project using Database & Data Structures Concepts

## Git Bash Commands:

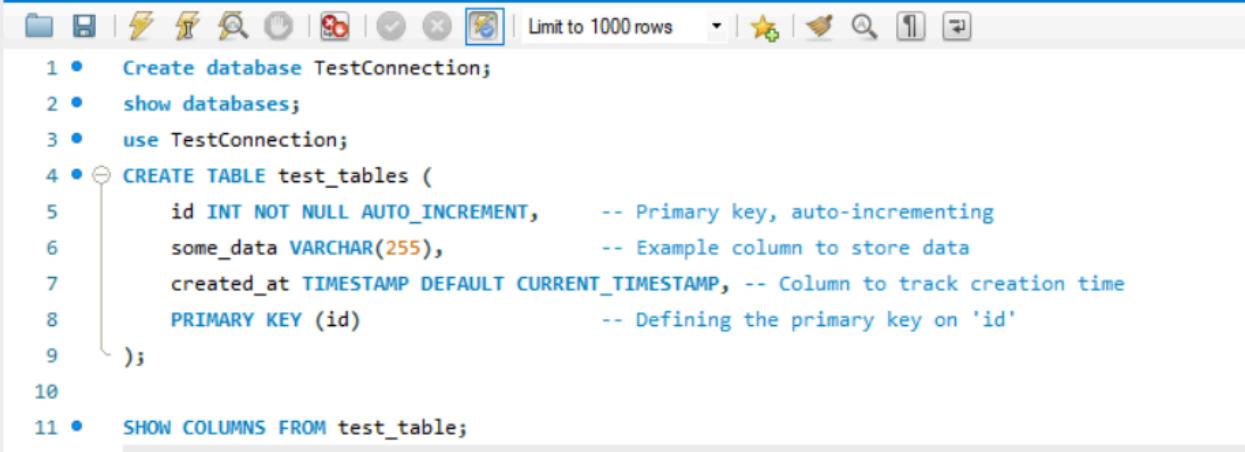
```
Anjali@Anjali MINGW64 ~ (new-feature)
$ cd C:\Codebase

Anjali@Anjali MINGW64 /c/Codebase (main)
$ mkdir AdvancedCaching

Anjali@Anjali MINGW64 /c/Codebase (main)
$ cd AdvancedCaching
```

```
Anjali@Anjali MINGW64 /c/Codebase/AdvancedCaching (main)
$ mvn archetype:generate \
  -DgroupId=com.example \
  -DartifactId=AdvancedCaching \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DinteractiveMode=false
MINGW support requires --add-opens java.base/java.lang=ALL-UNNAMED
MINGW support requires --add-opens java.base/java.lang=ALL-UNNAMED
[INFO] Scanning for projects...
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/3.2.0/maven-clean-plugin-3.2.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/3.2.0/maven-clean-plugin-3.2.0.pom (5.3 kB at 6.6 kB/s)
[INFO] Generating project in Batch mode
Downloading From central: https://repo.maven.apache.org/maven2/archetype-catalog.xml
Downloaded from central: https://repo.maven.apache.org/maven2/archetype-catalog.xml (16 kB at 703 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.jar (4.3 kB at 86 kB/s)
[INFO]
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO]
[INFO] Parameter: basedir, Value: C:\Codebase\AdvancedCaching
[INFO] Parameter: package, Value: com.example
[INFO] Parameter: groupId, Value: com.example
[INFO] Parameter: artifactId, Value: AdvancedCaching
[INFO] Parameter: packageName, Value: com.example
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Codebase\AdvancedCaching\AdvancedCaching
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 01:28 min
[INFO] Finished at: 2024-09-30T09:38:37+05:30
[INFO]
```

## Program 56 - Demonstrating Advanced Caching concepts using Database



The screenshot shows a MySQL Workbench interface with a toolbar at the top. Below the toolbar is a code editor containing the following SQL script:

```
1 • Create database TestConnection;
2 • show databases;
3 • use TestConnection;
4 • CREATE TABLE test_table (
5     id INT NOT NULL AUTO_INCREMENT,      -- Primary key, auto-incrementing
6     some_data VARCHAR(255),              -- Example column to store data
7     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- Column to track creation time
8     PRIMARY KEY (id)                  -- Defining the primary key on 'id'
9 );
10
11 • SHOW COLUMNS FROM test_table;
```

```

AdvancedCaching > AdvancedCaching > src > main > java > com > example > J AdvancedDatabaseCachingBenchmark.java > AdvancedDatabaseCachingBenchmark > DB_URL
1 package com.example;
2 import java.sql.Connection;
3 import java.util.Map;
4 import com.google.common.cache.Cache;
5 import java.sql.Statement;
6 import java.sql.SQLException;
7 import java.sql.DriverManager;
8 import java.util.concurrent.TimeUnit;
9 import java.util.LinkedHashMap;
10 import com.google.common.cache.CacheBuilder;
11 import java.sql.PreparedStatement;
12 import java.sql.ResultSet;
13
14 /**
15  * Hello world!
16  */
17
18 public class AdvancedDatabaseCachingBenchmark {
19 {
20     private static final String DB_URL = "jdbc:mysql://localhost:3306/TestConnection";
21     private static final String DB_USER = "root";
22     private static final String DB_PASSWORD = "root";
23     private static final int NUM_ELEMENTS = 100000;
24     private static final int L1_CACHE_SIZE = 10000;
25     private static final int L2_CACHE_SIZE = 10000;
26     private static final int L3_CACHE_SIZE = 100000;
27     private static final int L2_CACHE_DURATION_MINUTES = 10;
28
29     private static Connection connection;
30     private static Map<Integer, String> l1Cache;
31     private static Cache<Integer, String> l2Cache;
32
33     Run|Debug
34     public static void main( String[] args )
35     {
36         try{
37             setupDatabase();
38             setupCaches();
39             long dbInsertTime = benchmarkDatabaseInsert();
40             long dbRetrieveTime = benchmarkDatabaseRetrieve();
41             long l1CacheInsertTime = benchmarkL1CacheInsert();
42             long l1CacheRetrieveTime = benchmarkL1CacheRetrieve();
43             long l2CacheInsertTime = benchmarkL2CacheInsert();
44             long l2CacheRetrieveTime = benchmarkL2CacheRetrieve();
45             long multilevelCacheRetrieveTime = benchmarkMultilevelCacheRetrieve();
46             printResults(dbInsertTime, dbRetrieveTime, l1CacheInsertTime, l1CacheRetrieveTime, l2CacheInsertTime, l2CacheRetrieveTime, multilevelCacheRetrieveTime);
47
48         }catch(Exception e){
49             e.printStackTrace();
50         }finally{
51             try{
52                 connection.close();
53             }catch(SQLException e){
54                 e.printStackTrace();
55             }
56         }
57     }
58     private static void setupDatabase() {
59         try {
60             connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
61             Statement statement = connection.createStatement();
62             statement.executeUpdate("CREATE TABLE IF NOT EXISTS test_table (id INT PRIMARY KEY, value VARCHAR(255))");
63
64             statement.close();
65         } catch (SQLException e) {
66             e.printStackTrace();
67         }
68     }

```

```

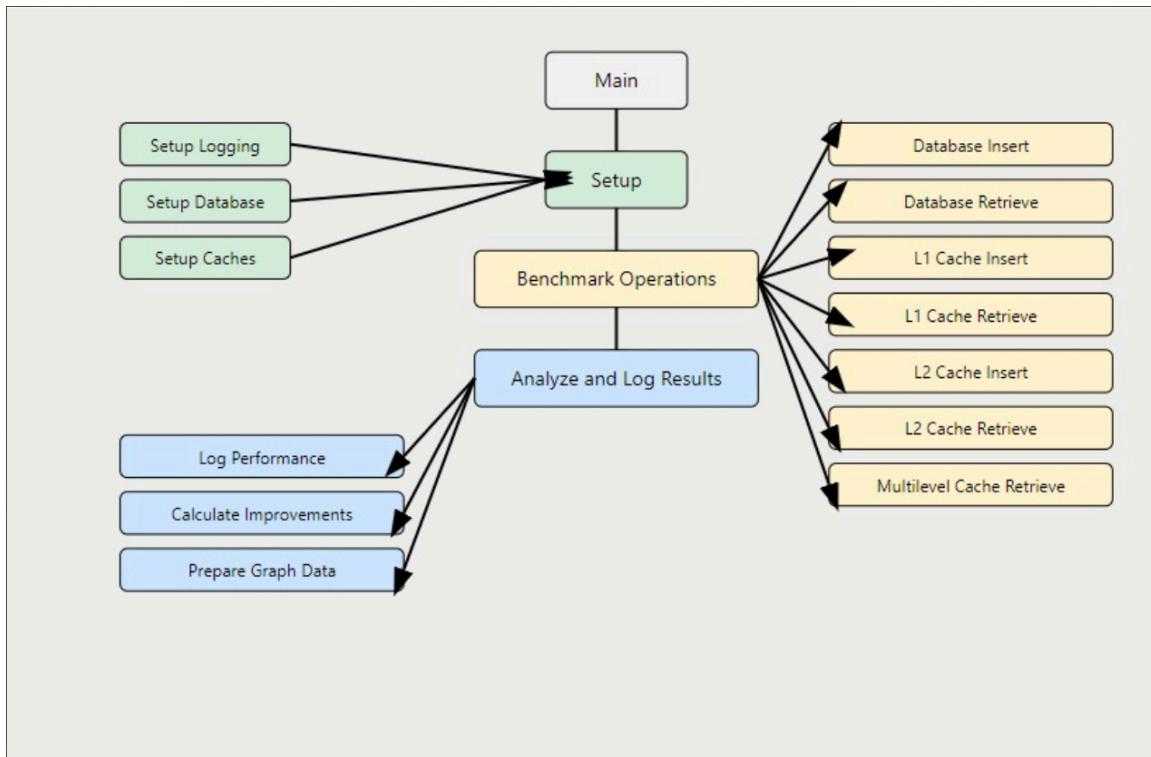
67 }
68 private static void setupCaches() {
69     l1Cache = new LinkedHashMap<Integer, String>(L1_CACHE_SIZE, 0.75f, true) {
70         @Override
71         protected boolean removeEldestEntry(Map.Entry<Integer, String> eldest) {
72             return size() > L1_CACHE_SIZE;
73         }
74     };
75     l2Cache = CacheBuilder.newBuilder()
76         .maximumSize(L2_CACHE_SIZE)
77         .expireAfterAccess(L2_CACHE_DURATION_MINUTES, TimeUnit.MINUTES)
78         .build();
79 }
80 private static long benchmarkDatabaseInsert() throws SQLException {
81     long startTime = System.nanoTime();
82     String sql = "INSERT INTO test_table (id, value) VALUES (?, ?)";
83     PreparedStatement statement = connection.prepareStatement(sql);
84     for (int i = 0; i < NUM_ELEMENTS; i++) {
85         statement.setInt(1, i);
86         statement.setString(2, "Value" + i);
87         statement.addBatch();
88         if(i %100 ==0){
89             statement.executeBatch();
90         }
91     }
92     statement.executeBatch();
93     statement.close();
94     long endTime = System.nanoTime();
95     return endTime - startTime;
96 }
97 private static long benchMarkDatabaseRetrieve() throws SQLException {
98     long startTime = System.nanoTime();
99     String sql = "SELECT * FROM test_table WHERE id = ?";
100    PreparedStatement statement = connection.prepareStatement(sql);
101    for (int i = 0; i < NUM_ELEMENTS; i++) {
102        statement.setInt(1, i);
103        ResultSet resultSet = statement.executeQuery();
104        if(resultSet.next()){
105            resultSet.getString("value");
106        }
107        resultSet.close();
108    }
109    statement.close();
110    long endTime = System.nanoTime();
111    return endTime - startTime;
112 }
113 private static long benchMarkL1CacheInsert()
114 {
115     long startTime = System.nanoTime();
116     for (int i = 0; i < NUM_ELEMENTS; i++) {
117         l1Cache.put(i, "Value" + i);
118     }
119     long endTime = System.nanoTime();
120     return endTime - startTime;
121 }
122 private static long benchMarkL1CacheRetrieve() {
123     long startTime = System.nanoTime();
124     for (int i = 0; i < NUM_ELEMENTS; i++) {
125         l1Cache.get(i);
126     }
127     long endTime = System.nanoTime();
128     return endTime - startTime;
129 }
130 private static long benchMarkL2CacheInsert() {

```

```

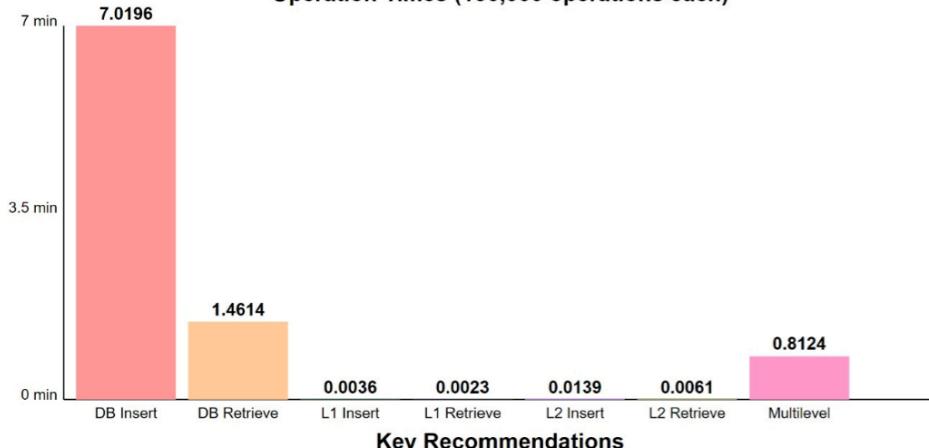
131     }
132     private static long benchMarkL2CacheInsert() {
133         long startTime = System.nanoTime();
134         for (int i = 0; i < NUM_ELEMENTS; i++) {
135             l2Cache.put(i, "Value" + i);
136         }
137         long endTime = System.nanoTime();
138         return endTime - startTime;
139     }
140     private static long benchMarkL2CacheRetrieve() {
141         long startTime = System.nanoTime();
142         for (int i = 0; i < NUM_ELEMENTS; i++) {
143             l2Cache.getIfPresent(i);
144         }
145         long endTime = System.nanoTime();
146         return endTime - startTime;
147     }
148     private static long benchmarkMultilevelCacheRetrieve() throws SQLException{
149         long startTime = System.nanoTime();
150         for (int i = 0; i < NUM_ELEMENTS; i++) {
151             String value = l1Cache.get(i);
152             if(value == null){
153                 value = l2Cache.getIfPresent(i);
154                 if(value == null){
155                     String sql = "SELECT * FROM test_table WHERE id = ?";
156                     PreparedStatement statement = connection.prepareStatement(sql);
157                     statement.setInt(1, i);
158                     ResultSet resultSet = statement.executeQuery();
159                     if(resultSet.next()){
160                         value = resultSet.getString("value");
161                         l2Cache.put(i, value);
162                         l1Cache.put(i, value);
163                     }
164                     resultSet.close();
165                     statement.close();
166                 }
167                 l1Cache.put(i, value);
168             }else{
169                 l2Cache.put(i, value);
170                 l1Cache.put(i, value);
171             }
172         }
173         long endTime = System.nanoTime();
174         return endTime - startTime;
175     }
176     private static void printResults(long dbInsertTime, long dbRetrieveTime,
177         long l1CacheInsertTime, long l1CacheRetrieveTime, long l2CacheInsertTime,
178         long l2CacheRetrieveTime, long multilevelCacheRetrieveTime){
179         System.out.println("Database Insert Time: " + dbInsertTime + " nanoseconds");
180         System.out.println("Database Retrieve Time: " + dbRetrieveTime + " nanoseconds");
181         System.out.println("L1 Cache Insert Time: " + l1CacheInsertTime + " nanoseconds");
182         System.out.println("L1 Cache Retrieve Time: " + l1CacheRetrieveTime + " nanoseconds");
183         System.out.println("L2 Cache Insert Time: " + l2CacheInsertTime + " nanoseconds");
184         System.out.println("L2 Cache Retrieve Time: " + l2CacheRetrieveTime + " nanoseconds");
185         System.out.println("Multilevel Cache Retrieve Time: " + multilevelCacheRetrieveTime + " nanoseconds");
186     }
187 }

```



## Advanced Database Caching Benchmark: Performance and Recommendations

Operation Times (100,000 operations each)



### Key Recommendations

1. Implement L1 Cache (LinkedHashMap) for critical, high-frequency data
2. Use L2 Cache (Guava) for larger, less frequent datasets
3. Deploy multilevel strategy for balanced performance
4. Optimize DB inserts to reduce 4.8x performance gap vs retrieval
5. Fine-tune cache sizes based on data access patterns
6. Consider async cache warming for multilevel strategy

### Key Insights

- L1 Cache: ~430x faster than DB retrieve, best for small, frequent data
- L2 Cache: ~240x faster than DB retrieve, excellent for larger datasets

## Program 57 - Benchmarking setup for concurrent database operations and caching

```
AdvancedCaching > AdvancedCaching > src > main > java > com > J P57AdvancedConcurrentDatabaseCachingBenchmark.java > P57AdvancedConcurrentDatabaseCachingBenchmark > BATCH_SIZE
1 import java.sql.Connection;
2 import java.util.Map;
3 import java.util.concurrent.*;
4 import java.util.function.Supplier;
5 import com.google.common.cache.Cache;
6 import java.sql.Statement;
7 import java.sql.SQLException;
8 import java.sql.DriverManager;
9 import java.util.LinkedHashMap;
10 import com.google.common.cache.CacheBuilder;
11 import java.sql.PreparedStatement;
12 import java.sql.ResultSet;
13 import java.util.logging.Logger;
14 import java.util.logging.Level;
15 import java.util.logging.FileHandler;
16 import java.util.logging.SimpleFormatter;
17 import java.util.ArrayList;
18 import java.util.List;
19 import java.util.Random;
20 import java.util.stream.IntStream;
21
22 public class P57AdvancedConcurrentDatabaseCachingBenchmark {
23     private static final String DB_URL = "jdbc:mysql://localhost:3306/TestConnection";
24     private static final String DB_USER = "root";
25     private static final String DB_PASSWORD = "root";
26     private static final int NUM_ELEMENTS = 100000;
27     private static final int L1_CACHE_SIZE = 10000;
28     private static final int L2_CACHE_SIZE = 10000;
29     private static final int L2_CACHE_DURATION_MINUTES = 10;
30     private static final int NUM_THREADS = Runtime.getRuntime().availableProcessors();
31     private static final int BATCH_SIZE = 10000;
32
33     private static Connection connection;
34     private static Map<Integer, String> l1Cache;
35     private static Cache<Integer, String> l2Cache;
36     private static final Logger LOGGER = Logger.getLogger(P57AdvancedConcurrentDatabaseCachingBenchmark.class.getName());
37     private static final ExecutorService executorService = Executors.newFixedThreadPool(NUM_THREADS);
38     private static final Random random = new Random();
39
40     Run | Debug
41     public static void main(String[] args) {
42         setupLogging();
43         try {
44             setupDatabase();
45             setupCaches();
46
47             List<BenchmarkResult> results = new ArrayList<>();
48             results.add(new BenchmarkResult(operation:"Database Insert", benchmarkConcurrent(P57AdvancedConcurrentDatabaseCachingBenchmark::databaseInsert)));
49             results.add(new BenchmarkResult(operation:"Database Retrieve", benchmarkConcurrent(P57AdvancedConcurrentDatabaseCachingBenchmark::databaseRetrieve)));
50             results.add(new BenchmarkResult(operation:"L1 Cache Insert", benchmarkConcurrent(P57AdvancedConcurrentDatabaseCachingBenchmark::l1CacheInsert)));
51             results.add(new BenchmarkResult(operation:"L1 Cache Retrieve", benchmarkConcurrent(P57AdvancedConcurrentDatabaseCachingBenchmark::l1CacheRetrieve)));
52             results.add(new BenchmarkResult(operation:"L2 Cache Insert", benchmarkConcurrent(P57AdvancedConcurrentDatabaseCachingBenchmark::l2CacheInsert)));
53             results.add(new BenchmarkResult(operation:"L2 Cache Retrieve", benchmarkConcurrent(P57AdvancedConcurrentDatabaseCachingBenchmark::l2CacheRetrieve)));
54             results.add(new BenchmarkResult(operation:"Multilevel Cache Retrieve", benchmarkConcurrent(P57AdvancedConcurrentDatabaseCachingBenchmark::multilevelCacheRetrieve)));
55         }
56     }
57 }
```

```

55     analyzeAndLogResults(results);
56
57 } catch (Exception e) {
58     LOGGER.log(Level.SEVERE, "An error occurred during benchmark execution", e);
59 } finally {
60     try {
61         if (connection != null) {
62             connection.close();
63         }
64         executorService.shutdown();
65     } catch (SQLException e) {
66         LOGGER.log(Level.SEVERE, "Error closing database connection", e);
67     }
68 }
69 }
70
71 private static void setupLogging() {
72     try {
73         FileHandler fileHandler = new FileHandler("concurrent_benchmark_log.txt");
74         SimpleFormatter formatter = new SimpleFormatter();
75         fileHandler.setFormatter(formatter);
76         LOGGER.addHandler(fileHandler);
77     } catch (Exception e) {
78         LOGGER.log(Level.SEVERE, "Error setting up logging", e);
79     }
80 }
81
82 private static void setupDatabase() {
83     try {
84         connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
85         Statement statement = connection.createStatement();
86         statement.executeUpdate("CREATE TABLE IF NOT EXISTS test_table (id INT PRIMARY KEY, value VARCHAR(255))");
87         statement.close();
88         LOGGER.info("Database setup completed successfully.");
89     } catch (SQLException e) {
90         LOGGER.log(Level.SEVERE, "Error setting up database", e);
91     }
92 }
93
94 private static void setupCaches() {
95     l1Cache = new ConcurrentHashMap<>(L1_CACHE_SIZE);
96     l2Cache = CacheBuilder.newBuilder()
97         .maximumSize(L2_CACHE_SIZE)
98         .expireAfterAccess(L2_CACHE_DURATION_MINUTES, TimeUnit.MINUTES)
99         .build();
100    LOGGER.info("Caches setup completed successfully.");
101 }
102
103 private static long benchmarkConcurrent(Supplier<Void> operation) throws InterruptedException, ExecutionException {
104     long startTime = System.nanoTime();
105     List<Future<Void>> futures = new ArrayList<*>();
106
107     for (int i = 0; i < NUM_THREADS; i++) {

```

```

105     private static long benchmarkConcurrent(Supplier<Void> operation) throws InterruptedException, ExecutionException {
106         for (int i = 0; i < NUM_THREADS; i++) {
107             int startIndex = i * (NUM_ELEMENTS / NUM_THREADS);
108             int endIndex = (i + 1) * (NUM_ELEMENTS / NUM_THREADS);
109             futures.add(executorService.submit(() -> {
110                 IntStream.range(startIndex, endIndex).forEach(j -> operation.get());
111                 return null;
112             }));
113         }
114     }
115
116     for (Future<Void> future : futures) {
117         future.get();
118     }
119
120     long endTime = System.nanoTime();
121     return endTime - startTime;
122 }
123
124 private static Void databaseInsert() {
125     try {
126         String sql = "INSERT INTO test_table (id, value) VALUES (?, ?) ON DUPLICATE KEY UPDATE value = VALUES(value)";
127         PreparedStatement statement = connection.prepareStatement(sql);
128         int id = random.nextInt(NUM_ELEMENTS);
129         statement.setInt(1, id);
130         statement.setString(2, "Value" + id);
131         statement.executeUpdate();
132         statement.close();
133     } catch (SQLException e) {
134         LOGGER.log(Level.SEVERE, "Error in database insert", e);
135     }
136     return null;
137 }
138
139 private static Void databaseRetrieve() {
140     try {
141         String sql = "SELECT value FROM test_table WHERE id = ?";
142         PreparedStatement statement = connection.prepareStatement(sql);
143         int id = random.nextInt(NUM_ELEMENTS);
144         statement.setInt(1, id);
145         ResultSet resultSet = statement.executeQuery();
146         if (resultSet.next()) {
147             resultSet.getString("value");
148         }
149         resultSet.close();
150         statement.close();
151     } catch (SQLException e) {
152         LOGGER.log(Level.SEVERE, "Error in database retrieve", e);
153     }
154     return null;
155 }
156
157 private static Void l1CacheInsert() {
158     int id = random.nextInt(NUM_ELEMENTS);
159     l1Cache.put(id, "Value" + id);

```

```

169
170     private static Void l2CacheInsert() {
171         int id = random.nextInt(NUM_ELEMENTS);
172         l2Cache.put(id, "Value" + id);
173         return null;
174     }
175
176     private static Void l2CacheRetrieve() {
177         int id = random.nextInt(NUM_ELEMENTS);
178         l2Cache.getIfPresent(id);
179         return null;
180     }
181
182     private static Void multilevelCacheRetrieve() {
183         int id = random.nextInt(NUM_ELEMENTS);
184         String value = l1Cache.get(id);
185         if (value == null) {
186             value = l2Cache.getIfPresent(id);
187             if (value == null) {
188                 try {
189                     String sql = "SELECT value FROM test_table WHERE id = ?";
190                     PreparedStatement statement = connection.prepareStatement(sql);
191                     statement.setInt(1, id);
192                     ResultSet resultSet = statement.executeQuery();
193                     if (resultSet.next()) {
194                         value = resultSet.getString("value");
195                         l2Cache.put(id, value);
196                         l1Cache.put(id, value);
197                     }
198                     resultSet.close();
199                     statement.close();
200                 } catch (SQLException e) {
201                     LOGGER.log(Level.SEVERE, "Error in multilevel cache retrieve", e);
202                 }
203             } else {
204                 l1Cache.put(id, value);
205             }
206         }
207         return null;
208     }
209
210     private static void analyzeAndLogResults(List<BenchmarkResult> results) {
211         LOGGER.info("Performance Analysis:");
212         for (BenchmarkResult result : results) {
213             logPerformance(result.operation, result.time);
214         }
215
216         LOGGER.info("\nPerformance Improvements:");
217         BenchmarkResult dbRetrieve = results.stream().filter(r -> r.operation.equals("Database Retrieve")).findFirst().orElseThrow();
218         results.stream()
219             .filter(r -> !r.operation.equals("Database Retrieve") && r.operation.equals("Database Insert"))
220             .forEach(r -> logImprovement(r.operation + " vs Database Retrieve", dbRetrieve.time, r.time));
221
222         identifyBottlenecks(results);
223     }
224
225     private static void logPerformance(String operation, long time) {
226         LOGGER.info(String.format("%s: %.2f ms", operation, time / 1_000_000.0));
227     }
228
229     private static void logImprovement(String comparison, long baseTime, long improvedTime) {
230         double improvement = (baseTime - improvedTime) / (double) baseTime * 100;
231         LOGGER.info(String.format("%s: %.2f%% faster", comparison, improvement));
232     }
233
234     private static void identifyBottlenecks(List<BenchmarkResult> results) {
235         BenchmarkResult slowest = results.stream().max((a, b) -> Long.compare(a.time, b.time)).orElseThrow();
236         LOGGER.warning("\nBottleneck: " + slowest.operation + " is the slowest operation.");
237     }
238
239     private static class BenchmarkResult {
240         String operation;
241         long time;
242
243         BenchmarkResult(String operation, long time) {
244             this.operation = operation;
245             this.time = time;
246         }
247     }

```

## Program 58 - Advanced Data Structures using Trie Map

```
AdvancedCaching > AdvancedCaching > src > main > java > com > example > P58AdvancedDataStructures.java > P58AdvancedDataStructures
 1 package com.example;
 2 import java.util.Map;
 3 import java.util.*;
 4 public class P58AdvancedDataStructures {
 5     /*
 6      * Why trie map?
 7      * TrieMap is a data structure that is used to store a dynamic set of strings.
 8      * It is a tree-like structure that is used to store strings in a way that allows for efficient prefix search.
 9      * It is also known as a prefix tree.
10      * It is used in many applications such as spell checking, autocomplete, and routing.
11      * It is also used in many databases such as Redis.
12      * It is also used in many search engines such as Elasticsearch.
13      * It is also used in many routing algorithms such as Dijkstra's algorithm.
14      * It is also used in many compression algorithms such as Huffman coding.
15      * It is also used in many encryption algorithms such as AES.
16      */
17     private static void demonstrateTrieMap(){
18         System.out.println("Demonstrating TrieMap");
19         TrieMap<Integer> trieMap = new TrieMap<>();
20         trieMap.put(key:"apple", value:1);
21         trieMap.put(key:"banana", value:2);
22         trieMap.put(key:"orange", value:3);
23         // System.out.println("TrieMap size: " + trieMap.size());
24         System.out.println("TrieMap contains key 'banana': " + trieMap.get(key:"apple"));
25         System.out.println("TrieMap contains key 'grape': " + trieMap.get(key:"grape"));
26         //System.out.println("TrieMap prefix search for 'app': " + trieMap.prefixSearch("app"));
27         //System.out.println("TrieMap prefix search for 'ora': " + trieMap.prefixSearch("ora"));
28         //System.out.println("TrieMap keys: " + trieMap.keys());
29         //System.out.println("TrieMap values: " + trieMap.values());
30     }
31     Run | Debug
32     public static void main(String[] args) {
33         demonstrateTrieMap();
34     }
35 }
36 class TrieNode<T> {
37     Map<Character, TrieNode<T>> children = new HashMap<>();
38     T value;
39     boolean isEndOfWord;
40 }
41 class TrieMap<T> {
42     private TrieNode<T> root =new TrieNode<>();
43
44     public void put(String key, T value){
45         TrieNode<T> current = root;
46         for(char c : key.toCharArray()){


```

```

47     |         current.children.computeIfAbsent(c, t -> new TrieNode<T>());
48   }
49   current.value = value;
50   current.isEndOfWord = true;
51 }
52 public T get(String key){
53     TrieNode<T> current = findNode(key);
54     return current != null && current.isEndOfWord ? current.value : null;
55 }
56 private TrieNode<T> findNode(String key){
57     TrieNode<T> current = root;
58     for(char c : key.toCharArray()){
59         current = current.children.get(c);
60         if(current == null){
61             return null;
62         }
63     }
64     return current;
65 }
66 public boolean containsKey(String key){
67     return get(key) != null;
68 }
69 public boolean hasPrefix(String prefix){
70     return findNode(prefix) != null;
71 }
72 public List<String> getWordsForPrefix(String prefix){
73     List<String> results = new ArrayList<String>();
74     TrieNode<T> node = findNode(prefix);
75     if(node != null){
76         collectAllWords(node, prefix, results);
77     }
78     return results;
79 }
80 private void collectAllWords(TrieNode<T> node, String prefix, List<String> results){
81     if(node.isEndOfWord){
82         results.add(prefix);
83     }
84     for(Map.Entry<Character, TrieNode<T> > entry : node.children.entrySet()){
85         collectAllWords(entry.getValue(), prefix + entry.getKey(), results);
86     }
87 }
88 }

```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Demostrating TrieMap  
TrieMap contains key 'banana': null  
TrieMap contains key 'grape': null  
TrieMap contains key 'grape': null  
PS C:\Codebase>

## Program 59 - Demonstration of Trie

```
AdvancedCaching > AdvancedCaching > src > main > java > com > example > P59TrieDemo.java > P59TrieDemo
1 package com.example;
2
3 import java.util.*;
4
5 // TrieNode class represents a single node in the Trie
6 class TrieNode<T> {
7     // Map to store child nodes, where the key is the character and value is the child TrieNode
8     Map<Character, TrieNode<T>> children;
9
10    // Value associated with this node (null if this node doesn't represent the end of a word)
11    T value;
12
13    // Flag to indicate if this node represents the end of a word
14    boolean isEndOfWord;
15
16    // Constructor to initialize a new TrieNode
17    public TrieNode() {
18        this.children = new HashMap<>();
19        this.value = null;
20        this.isEndOfWord = false;
21    }
22}
23
24 // Trie class implements the main functionality of the Trie data structure
25 class Trie<T> {
26     // Root node of the Trie
27     private TrieNode<T> root;
28
29     // Constructor to initialize an empty Trie
30     public Trie() {
31         this.root = new TrieNode<T>();
32     }
33
34     // Method to insert a word and its associated value into the Trie
35     public void insert(String word, T value) {
36         TrieNode<T> current = root;
37
38         // Iterate through each character in the word
39         for (char c : word.toCharArray()) {
40             // If the current character doesn't exist as a child, create a new node
41             current = current.children.computeIfAbsent(c, k -> new TrieNode<T>());
42         }
43
44         // Mark the last node as the end of a word and set its value
45         current.isEndOfWord = true;
46         current.value = value;
47     }
48
49     // Method to search for a word in the Trie
50     public T search(String word) {
51         TrieNode<T> node = findNode(word);
52         // Return the value if the word exists and is marked as end of word, otherwise null
53         return (node != null && node.isEndOfWord) ? node.value : null;
54     }
55
56     // Method to check if any word in the Trie starts with the given prefix
57     public boolean startsWith(String prefix) {
58         // If we can find a node for this prefix, it exists in the Trie
59         return findNode(prefix) != null;
60     }
61
62     // Helper method to find a node corresponding to a given word or prefix
63     private TrieNode<T> findNode(String str) {
64         TrieNode<T> current = root;
65
66         // Traverse the Trie following the characters in the string
67         for (char c : str.toCharArray()) {
```

```

68     current = current.children.get(c);
69     // If at any point we can't find a child node, the word/prefix doesn't exist
70     if (current == null) return null;
71   }
72
73   // Return the final node we reached
74   return current;
75 }
76
77 // Method to get all words in the Trie with a given prefix
78 public List<String> getWordsWithPrefix(String prefix) {
79   List<String> result = new ArrayList<>();
80   TrieNode<T> prefixNode = findNode(prefix);
81
82   // If the prefix exists in the Trie, collect all words starting from its last node
83   if (prefixNode != null) {
84     collectWords(prefixNode, prefix, result);
85   }
86
87   return result;
88 }
89
90 // Recursive helper method to collect all words from a given node
91 private void collectWords(TrieNode<T> node, String currentPrefix, List<String> result) {
92   // If this node is marked as end of word, add the current prefix to results
93   if (node.isEndOfWord) {
94     result.add(currentPrefix);
95   }
96
97   // Recursively explore all child nodes
98   for (Map.Entry<Character, TrieNode<T>> entry : node.children.entrySet()) {
99     collectWords(entry.getValue(), currentPrefix + entry.getKey(), result);
100   }
101 }
102 }
103
104 // Main class to demonstrate the usage of the Trie
105 public class P59TrieDemo {
106   Run|Debug
107   public static void main(String[] args) {
108     Trie<Integer> trie = new Trie<>();
109
110     // Insert some words with associated values
111     trie.insert(word:"apple", value:1);
112     trie.insert(word:"app", value:2);
113     trie.insert(word:"application", value:3);
114     trie.insert(word:"banana", value:4);
115
116     // Demonstrate search functionality
117     System.out.println("Value for 'apple': " + trie.search(word:"apple")); // Output: 1
118     System.out.println("Value for 'app': " + trie.search(word:"app")); // Output: 2
119     System.out.println("Value for 'appl': " + trie.search(word:"appl")); // Output: null
120
121     // Demonstrate prefix checking
122     System.out.println("Starts with 'app': " + trie.startsWith(prefix:"app")); // Output: true
123     System.out.println("Starts with 'ban': " + trie.startsWith(prefix:"ban")); // Output: true
124     System.out.println("Starts with 'car': " + trie.startsWith(prefix:"car")); // Output: false
125
126     // Demonstrate getting words with a prefix
127     System.out.println("Words with prefix 'app': " + trie.getWordsWithPrefix(prefix:"app"));
128   }
129 }
130

```

```

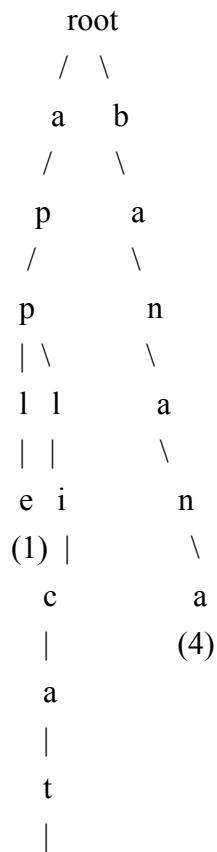
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Codebase> & 'C:\Program Files\Java\jre-1.8\bin\java.exe' '-cp' 'C:\Users\Anjali\AppData\Local\Temp\cp_e9zja26nd76n0w82mfr3mbpc.jar' 'com.example.P59TrieDemo'
Value for 'apple': 1
Value for 'app': 2
PS C:\Codebase> & 'C:\Program Files\Java\jre-1.8\bin\java.exe' '-cp' 'C:\Users\Anjali\AppData\Local\Temp\cp_e9zja26nd76n0w82mfr3mbpc.jar' 'com.example.P59TrieDemo'
Value for 'apple': 1
PS C:\Codebase> & 'C:\Program Files\Java\jre-1.8\bin\java.exe' '-cp' 'C:\Users\Anjali\AppData\Local\Temp\cp_e9zja26nd76n0w82mfr3mbpc.jar' 'com.example.P59TrieDemo'
PS C:\Codebase> & 'C:\Program Files\Java\jre-1.8\bin\java.exe' '-cp' 'C:\Users\Anjali\AppData\Local\Temp\cp_e9zja26nd76n0w82mfr3mbpc.jar' 'com.example.P59TrieDemo'
Value for 'apple': 1
Value for 'app': 2
Value for 'appl': null
Starts with 'app': true
Starts with 'ban': true
Starts with 'car': false
Value for 'appl': null
Starts with 'app': true
Starts with 'ban': true
Starts with 'car': false
Starts with 'ban': true
Starts with 'car': false
Words with prefix 'app': [app, apple, application]
Words with prefix 'app': [app, apple, application]
PS C:\Codebase>

```

## Trie Concepts:

### 1. Trie Structure:

First, let's visualize the Trie structure after inserting the words "apple", "app", "application", and "banana":



i  
|  
o  
|  
n  
(3)  
(2)

Numbers in parentheses represent the values associated with complete words.

Now, let's go through each operation:

2. Insertion (insert method):

- Start from the root.
- For each character in the word, create a new node if it doesn't exist.
- Mark the last node as the end of a word and set its value.

Example: Inserting "apple" with value 1

root -> a -> p -> p -> l -> e (isEndOfWord = true, value = 1)

3. Search (search method):

- Start from the root.
- Follow the path for each character in the word.
- If we reach the end and it's marked as a word end, return the value; otherwise, return null.

Example: Searching for "app"

root -> a -> p -> p (isEndOfWord = true, value = 2)

Return: 2

4. Prefix Check (startsWith method):

- Similar to search, but only checks if the prefix exists.
- Return true if we can follow the path for the entire prefix.

Example: Checking for prefix "ban"

root -> b -> a -> n (exists, so return true)

---

## 5. Get Words with Prefix (getWordsWithPrefix method):

- Find the node corresponding to the prefix.
- From that node, perform a depth-first search to collect all words.

Example: Getting words with prefix "app"

Start at: root -> a -> p -> p

Collect: "app" (value 2)

"apple" (value 1)

"application" (value 3)

## 6. Debug Operations:

To debug these operations, you can add print statements or use a debugger to observe:

a) Node creation during insertion:

java

```
System.out.println("Creating node for character: " + c);
```

b) Path traversal during search and startsWith:

java

```
System.out.println("Traversing node: " + c);
```

c) Word collection during getWordsWithPrefix:

java

```
System.out.println("Collecting word: " + currentPrefix);
```

## 7. Step-by-Step Debugging:

For example, let's debug the search for "apple":

Searching for "apple"

Traversing node: a

Traversing node: p

Traversing node: p

Traversing node: l

Traversing node: e

Found word "apple" with value 1

To visually debug, you could implement a method to print the Trie structure:

java

```
public void printTrie() {
    printTrieNode(root, "", "");
}

private void printTrieNode(TrieNode<T> node, String prefix, String childPrefix) {
    System.out.println(prefix + (node.isEndOfWord ? "* " : " ") + (node.value != null ? "(" +
node.value + ")" : ""));
    for (Map.Entry<Character, TrieNode<T>> entry : node.children.entrySet()) {
        printTrieNode(entry.getValue(), childPrefix + " |--- " + entry.getKey(), childPrefix + "|");
    }
}
```

This would produce a visual representation of the Trie structure, helping in debugging and understanding the current state of the Trie.

By adding these debug operations and visualizations, you can step through each operation and observe how the Trie is constructed and traversed, making it easier to understand and debug the code.

## Program 60 - Binary Search

```
Java > Code > P60BinarySearch.java > P60BinarySearch
1 import java.util.*;
2
3 public class P60BinarySearch {
4
5     public static int binarySearch(int[] nums, int target) {
6         int n = nums.length; //size of the array.
7         int low = 0, high = n - 1;
8
9         // Perform the steps:
10        while (low <= high) {
11            int mid = (low + high) / 2;
12            if (nums[mid] == target) return mid;
13            else if (target > nums[mid]) low = mid + 1;
14            else high = mid - 1;
15        }
16        return -1;
17    }
18
19    public static void main(String[] args) {
20        int[] a = {3, 4, 6, 7, 9, 12, 16, 17};
21        int target = 6;
22        int ind = binarySearch(a, target);
23        if (ind == -1)
24            System.out.println("The target is not present.");
25        else
26            System.out.println("The target is at index: " + ind);
27    }
28 }
```

PROBLEMS 9    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS C:\Codebase> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionTraces' 'P60BinarySearch'
The target is at index: 2
PS C:\Codebase>
```

## Program 61 - Binary Search using Recursion

```
Java > Code > P61BinaryRecursion.java > P61BinaryRecursion
1 import java.util.*;
2
3 public class P61BinaryRecursion {
4
5     // Recursive binary search function
6     public static int binarySearch(int[] nums, int target, int low, int high) {
7         if (low > high) {
8             return -1; // base case: target not found
9         }
10
11         int mid = (low + high) / 2;
12
13         // Check if target is present at mid
14         if (nums[mid] == target) {
15             return mid;
16         }
17
18         // If target is smaller than mid, search in the left subarray
19         if (target < nums[mid]) {
20             return binarySearch(nums, target, low, mid - 1);
21         }
22
23         // Else, search in the right subarray
24         return binarySearch(nums, target, mid + 1, high);
25     }
26
27     Run | Debug
28     public static void main(String[] args) {
29         int[] a = {3, 4, 6, 7, 9, 12, 16, 17};
30         int target = 6;
31
32         // Initial call to recursive binary search
33         int ind = binarySearch(a, target, low:0, a.length - 1);
34
35         if (ind == -1)
36             System.out.println("The target is not present.");
37         else
38             System.out.println("The target is at index: " + ind);
39     }
}
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessage\r\workspaceStorage\dc86b2a86068cf587cb193dcf15e39aa\redhat.java\jdt_ws\jdt.ls-java-project\bin' 'P61
The target is at index: 2
```

## Day 12 - Sorting and Working on LMS and Hackerearth

### Program 62 - Merge Sort

```
Java > Code > J P62MergeSort.java > ...
1  // MergeSort implementation in Java
2  public class MergeSort {
3
4      // Method to merge two subarrays
5      public static void merge(int[] array, int left, int middle, int right) {
6          int n1 = middle - left + 1;
7          int n2 = right - middle;
8
9          // Temporary arrays
10         int[] leftArray = new int[n1];
11         int[] rightArray = new int[n2];
12
13         // Copy data to temp arrays
14         for (int i = 0; i < n1; ++i)
15             leftArray[i] = array[left + i];
16         for (int j = 0; j < n2; ++j)
17             rightArray[j] = array[middle + 1 + j];
18
19         // Merge the temp arrays
20
21         // Initial indexes of the two subarrays
22         int i = 0, j = 0;
23
24         // Initial index of merged subarray
25         int k = left;
26         while (i < n1 && j < n2) {
27             if (leftArray[i] <= rightArray[j]) {
28                 array[k] = leftArray[i];
29                 i++;
30             } else {
31                 array[k] = rightArray[j];
32                 j++;
33             }
34             k++;
35         }
36
37         // Copy remaining elements of leftArray[] if any
38         while (i < n1) {
39             array[k] = leftArray[i];
40             i++;
41             k++;
42         }
43
44         // Copy remaining elements of rightArray[] if any
45         while (j < n2) {
46             array[k] = rightArray[j];
47             j++;
48             k++;
49         }
50     }
51
52     // Main function that sorts array[left...right] using merge()
53     public static void sort(int[] array, int left, int right) {
54         if (left < right) {
55             // Find the middle point
```

```
53     public static void sort(int[] array, int left, int right) {
54         if (left < right) {
55             // Find the middle point
56             int middle = (left + right) / 2;
57
58             // Sort first and second halves
59             sort(array, left, middle);
60             sort(array, middle + 1, right);
61
62             // Merge the sorted halves
63             merge(array, left, middle, right);
64         }
65     }
66
67     // Utility method to print the array
68     public static void printArray(int[] array) {
69         int n = array.length;
70         for (int i = 0; i < n; ++i)
71             System.out.print(array[i] + " ");
72         System.out.println();
73     }
74
75     // Main method to test the algorithm
76     Run | Debug
77     public static void main(String[] args) {
78         int[] array = {12, 11, 13, 5, 6, 7};
79         System.out.println("Given Array:");
80         printArray(array);
81
82         // Sorting the array using Merge Sort
83         sort(array, left:0, array.length - 1);
84
85         System.out.println("\nSorted Array:");
86         printArray(array);
87     }
88 }
89 Ctrl+L to chat, Ctrl+K to generate
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
5e39aa\redhat.java\jdt_ws\jdt.ls-java-project\bin' 'MergeSort'
Given Array:
12 11 13 5 6 7

Sorted Array:
5 6 7 11 12 13
PS C:\Codebase>
```

## Program 63 - Quick Sort

```
Java > Code > P63QuickSort.java > P63QuickSort
 1  public class P63QuickSort {
 2
 3      // Method to perform quicksort
 4      public static void quickSort(int[] arr, int low, int high) {
 5          if (low < high) {
 6              // Partition the array and get the pivot index
 7              int pi = partition(arr, low, high);
 8
 9              // Recursively sort the elements before and after partition
10              quickSort(arr, low, pi - 1);
11              quickSort(arr, pi + 1, high);
12          }
13      }
14
15      // Method to partition the array
16      public static int partition(int[] arr, int low, int high) {
17          // Choose the rightmost element as the pivot
18          int pivot = arr[high];
19          int i = low - 1; // Pointer for the greater element
20
21          // Traverse through all elements
22          for (int j = low; j < high; j++) {
23              // If the current element is smaller than or equal to the pivot
24              if (arr[j] <= pivot) {
25                  i++; // Increment the pointer
26                  // Swap arr[i] and arr[j]
27                  int temp = arr[i];
28                  arr[i] = arr[j];
29                  arr[j] = temp;
30              }
31          }
32
33          // Swap the pivot element with the element at i+1 position
34          int temp = arr[i + 1];
35          arr[i + 1] = arr[high];
36          arr[high] = temp;
37
38          return i + 1; // Return the pivot index
39      }
40
41      // Utility method to print the array
42      public static void printArray(int[] arr) {
43          for (int num : arr) {
44              System.out.print(num + " ");
45          }
46          System.out.println();
47      }
48
49      // Main method to test the Quick Sort
50      Run|Debug
51      public static void main(String[] args) {
52          int[] arr = {10, 80, 30, 90, 40, 50, 70};
53          System.out.println("Original Array:");
54          printArray(arr);
55
56          quickSort(arr, low:0, arr.length - 1);
57
58          System.out.println("\nSorted Array:");
59      }
}
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Codebase> & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\A
Original Array:
10 80 30 90 40 50 70

Sorted Array:
10 30 40 50 70 80 90
```

**A detailed explanation of the QuickSort algorithm:** focusing on the role of swapping and where the sorting happens. Let's go through the code step-by-step, using the example array [10, 7, 8, 9, 1, 5].

1. The main method initiates the sorting process:

- It creates an instance of QuickSort and calls ob.sort(arr, 0, 5).

2. The sort method is recursive and works as follows:

- It first checks if low < high. If true, it proceeds; otherwise, it returns (base case for recursion).
- It calls partition(arr, 0, 5) to get the partition index.

3. The partition method is where the main work happens:

- It chooses the last element (5) as the pivot.
- It initializes i to -1 (index of the smaller element).
- It iterates through the array from index 0 to 4.

Let's debug the partition method step-by-step:

Initial array: [10, 7, 8, 9, 1, 5]

Pivot: 5

i: -1

- j = 0: arr[0] = 10 > pivot, no swap
- j = 1: arr[1] = 7 > pivot, no swap
- j = 2: arr[2] = 8 > pivot, no swap
- j = 3: arr[3] = 9 > pivot, no swap
- j = 4: arr[4] = 1 < pivot
  - Increment i to 0
  - Swap arr[0] and arr[4]
  - Array becomes: [1, 7, 8, 9, 10, 5]

After the loop:

- Swap arr[i+1] (which is arr[1] = 7) with arr[high] (pivot = 5)

- 
- Array becomes: [1, 5, 8, 9, 10, 7]

The partition method returns 1 ( $i+1$ ), which becomes the partition index ( $pi$ ) in the sort method.

#### 4. Back in the sort method:

- It recursively calls  $\text{sort}(\text{arr}, 0, 0)$  for the left part (which immediately returns as  $0 < 0$ )
- It recursively calls  $\text{sort}(\text{arr}, 2, 5)$  for the right part

#### 5. For $\text{sort}(\text{arr}, 2, 5)$ :

- It calls  $\text{partition}(\text{arr}, 2, 5)$

Let's debug this partition call:

Current array: [1, 5, 8, 9, 10, 7]

Pivot: 7

$i: 1$

- $j = 2$ :  $\text{arr}[2] = 8 > \text{pivot}$ , no swap
- $j = 3$ :  $\text{arr}[3] = 9 > \text{pivot}$ , no swap
- $j = 4$ :  $\text{arr}[4] = 10 > \text{pivot}$ , no swap

After the loop:

- Swap  $\text{arr}[i+1]$  (which is  $\text{arr}[2] = 8$ ) with  $\text{arr}[\text{high}]$  ( $\text{pivot} = 7$ )
- Array becomes: [1, 5, 7, 9, 10, 8]

The partition method returns 2.

#### 6. Back in the sort method:

- It recursively calls  $\text{sort}(\text{arr}, 2, 1)$  for the left part (which immediately returns as  $2 < 1$ )
- It recursively calls  $\text{sort}(\text{arr}, 3, 5)$  for the right part

This process continues recursively until the entire array is sorted.

The role of swapping:

1. In the partition method, swapping serves two purposes:
  - a. It moves elements smaller than the pivot to the left side of the array.
  - b. It places the pivot in its final sorted position.
2. The swaps within the loop ( $\text{arr}[i]$  and  $\text{arr}[j]$ ) group elements smaller than the pivot together.
3. The final swap ( $\text{arr}[i+1]$  and  $\text{arr}[\text{high}]$ ) puts the pivot in its correct position, with all smaller elements to its left and all larger elements to its right.

Where sorting happens:

The sorting in QuickSort occurs through the combination of partitioning and recursion:

1. Each partition operation puts one element (the pivot) in its correct sorted position.
2. The recursive calls to sort then apply this process to increasingly smaller subarrays.
3. As the recursion deepens, more elements are put into their correct positions.
4. The sorting is complete when all subarrays have been partitioned.

In essence, QuickSort sorts by repeatedly partitioning the array and ensuring that each partition puts at least one element in its final sorted position. The recursion ensures that this process is applied to all elements in the array.

## Program 64 - Binary Search Tree

```
Java > Code > P64BinaryTree.java > P64BinaryTree > main(String[])
1  public class P64BinaryTree {
2      // Node class representing each node in the binary tree
3      class Node {
4          int data;
5          Node left, right;
6
7          public Node(int item) {
8              data = item;
9              left = right = null;
10         }
11     }
12
13     // Root of Binary Tree
14     Node root;
15
16     // Constructor
17     BinaryTree() {
18         root = null;
19     }
20
21     // Method to insert a new node
22     void insert(int key) {
23         root = insertRec(root, key);
24     }
25
26     // A recursive function to insert a new key in BST
27     Node insertRec(Node root, int key) {
28         // If the tree is empty, return a new node
29         if (root == null) {
30             root = new Node(key);
31             return root;
32         }
33
34         // Otherwise, recur down the tree
35         if (key < root.data)
36             root.left = insertRec(root.left, key);
37         else if (key > root.data)
38             root.right = insertRec(root.right, key);
39
40         // Return the (unchanged) node pointer
41         return root;
42     }
43
44     // Method to do inorder traversal of tree
45     void inorder() {
46         inorderRec(root);
```

```

50     void inorderRec(Node root) {
51         inorderRec(root.left);
52         System.out.print(root.data + " ");
53         inorderRec(root.right);
54     }
55 }
56
57
58 // Method to search a key in BST
59 boolean search(int key) {
60     return searchRec(root, key);
61 }
62
63
64 // A recursive function to search a given key in BST
65 boolean searchRec(Node root, int key) {
66     // Base Cases: root is null or key is present at root
67     if (root == null || root.data == key)
68         return root != null;
69
70     // Key is greater than root's key
71     if (root.data < key)
72         return searchRec(root.right, key);
73
74     // Key is smaller than root's key
75     return searchRec(root.left, key);
76 }
77
78 // Depth First Search (DFS) implementation
79 void depthFirstSearch() {
80     System.out.println("Depth First Search (Pre-order traversal):");
81     dfsRec(root);
82     System.out.println();
83 }
84
85 // A utility function to do DFS of BST
86 void dfsRec(Node node) {
87     if (node == null)
88         return;
89
90     // First print data of node
91     System.out.print(node.data + " ");
92
93     // Then recur on left subtree
94     dfsRec(node.left);
95
96     // Now recur on right subtree
97     dfsRec(node.right);
98 }
99
100 // Driver Code
101 Run | Debug
102 public static void main(String[] args) {
103     P64BinaryTree tree = new P64BinaryTree();
104
105     /* Let's create following BST
106      *          50
107      *        /   \
108      *      30   70
109      *     / \ / \
110      *    20 40 60 80
111      *   / \   / \
112      *  10 35 55 75
113      */
114 }
```

```

100
● 101     public static void main(String[] args) {
102
103         /* Let's create following BST
104             |
105             50
106             /   \
107            30   70
108           / \   / \
109          20 40 60 80 */
110         tree.insert(key:50);
111         tree.insert(key:30);
112         tree.insert(key:20);
113         tree.insert(key:40);
114         tree.insert(key:70);
115         tree.insert(key:60);
116         tree.insert(key:80);
117
118         // Print inorder traversal of the BST
119         System.out.println("Inorder traversal of the binary tree:");
120         tree.inorder();
121         System.out.println();
122
123         // Search for a key
124         int key = 60;
125         System.out.println("\nSearching for key " + key + ": " +
126                           (tree.search(key) ? "Found" : "Not Found"));
127
128         // Perform DFS
129         tree.depthFirstSearch();
130     }

```

PROBLEMS 13 OUTPUT DEBUG CONSOLE TERMINAL PORTS

5e39aa\redhat.java\jdt\_ws\jdt.ls-java-project\bin' 'P64BinaryTree'
Inorder traversal of the binary tree:
20 30 40 50 60 70 80

Searching for key 60: Found
Depth First Search (Pre-order traversal):
50 30 20 40 70 60 80
PS C:\Codebase>

PS C:\Codebase>

## Program 65 - Deleting a Key in Binary Search Tree

```
Java > Code > J P65DeleteKey.java > BinarySearchTree > main(String[])
 1  class Node {
 2     int key;
 3     Node left, right;
 4
 5     Node(int item) {
 6         key = item;
 7         left = right = null;
 8     }
 9 }
10
11 class BinarySearchTree {
12     Node root;
13
14     // Constructor
15     BinarySearchTree() {
16         root = null;
17     }
18
19     // Insert a value into the BST
20     void insert(int key) {
21         root = insertRec(root, key);
22     }
23
24     Node insertRec(Node root, int key) {
25         if (root == null) {
26             return new Node(key);
27         }
28         if (key < root.key) {
29             root.left = insertRec(root.left, key);
30         } else if (key > root.key) {
31             root.right = insertRec(root.right, key);
32         }
33         return root;
34     }
35
36     // Delete a key
37     void deleteKey(int key) {
38         root = deleteRec(root, key);
39     }
40
41     Node deleteRec(Node root, int key) {
42         // Base case: If the tree is empty
43         if (root == null) return root;
44
45         // Recursive calls for ancestors of node to be deleted
46         if (key < root.key) {
47             root.left = deleteRec(root.left, key);
48         } else if (key > root.key) {
49             root.right = deleteRec(root.right, key);
50         } else { // We reach here when root is the node to be deleted.
51             // Case 1: Node with only one child or no child
52             if (root.left == null) {
53                 return root.right;
54             } else if (root.right == null) {
55                 return root.left;
56             }
57
58             // Case 2: Node with two children
59             // Get the inorder successor (smallest in the right subtree)
60             root.key = minValue(root.right);
61
62             // Delete the inorder successor
63             root.right = deleteRec(root.right, root.key);
64         }
65         return root;
66     }
67 }
```

```

42     }
43 
44     Node deleteNode(Node root, int key) {
45         if (root == null)
46             return null;
47 
48         if (key < root.key)
49             root.left = deleteNode(root.left, key);
50         else if (key > root.key)
51             root.right = deleteNode(root.right, key);
52         else {
53             if (root.left == null)
54                 return root.right;
55             else if (root.right == null)
56                 return root.left;
57 
58             Node minNode = minValue(root.right);
59             root.key = minNode.key;
60             root.right = deleteNode(root.right, minNode.key);
61         }
62         return root;
63     }
64 
65     int minValue(Node root) {
66         int minv = root.key;
67         while (root.left != null) {
68             minv = root.left.key;
69             root = root.left;
70         }
71         return minv;
72     }
73 
74     // In-order traversal
75     void inOrderTraversal() {
76         inOrderRec(root);
77     }
78 
79     void inOrderRec(Node root) {
80         if (root != null) {
81             inOrderRec(root.left);
82             System.out.print(root.key + " ");
83             inOrderRec(root.right);
84         }
85     }
86 
87     // Main method to demonstrate functionality
88     Run | Debug
89     public static void main(String[] args) {
90         BinarySearchTree bst = new BinarySearchTree();
91         bst.insert(key:50);
92         bst.insert(key:30);
93         bst.insert(key:20);
94         bst.insert(key:40);
95         bst.insert(key:70);
96         bst.insert(key:60);
97         bst.insert(key:80);
98 
99         System.out.println("In-order traversal before deletion:");
100        bst.inOrderTraversal(); // Output: 20 30 40 50 60 70 80
101        System.out.println();
102 
103        System.out.println("Delete 20:");
104        bst.deleteKey(key:20);
105        bst.inOrderTraversal(); // Output: 30 40 50 60 70 80
106        System.out.println();
107 
108        System.out.println("Delete 30:");
109        bst.deleteKey(key:30);
110        bst.inOrderTraversal(); // Output: 40 50 60 70 80
111        System.out.println();
112 
113        System.out.println("Delete 50:");
114        bst.deleteKey(key:50);
115        bst.inOrderTraversal(); // Output: 40 60 70 80
116        System.out.println();
117    }
118 }
119 
120 }
121 
```

```
PROBLEMS 14 OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Codebase> & 'C:\Program Files\Java\jdk-17
In-order traversal before deletion:
20 30 40 50 60 70 80
Delete 20:
30 40 50 60 70 80
Delete 30:
40 50 60 70 80
Delete 50:
40 60 70 80
40 50 60 70 80
Delete 50:
40 60 70 80
PS C:\Codebase>
```

Certainly! Let's debug the delete operation step-by-step using the tree we created in the main method. We'll focus on the three deletion examples: deleting 20, 30, and 50.

Let's start with the initial tree:

```
      50
     /   \
    30   70
   / \   / \
  20 40 60 80
```

### 1. Deleting 20 (a leaf node):

When we call `tree.deleteKey(20)`, here's what happens:

- `deleteKey(20)` calls `deleteRec(root, 20)`
- In `deleteRec`:
  - $20 < 50$ , so we recurse to the left: `root.left = deleteRec(root.left, 20)`
  - $20 < 30$ , so we recurse to the left again: `root.left = deleteRec(root.left, 20)`
  - We find 20. It has no children, so we return null.
  - This null is assigned to the left child of 30.

After deletion, the tree looks like this:

```
      50
     /   \
    30   70
       \   \
      40 60 80
```

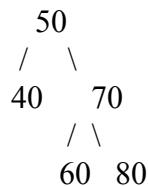
---

## 2. Deleting 30 (a node with one child):

When we call `tree.deleteKey(30)`, here's what happens:

- `deleteKey(30)` calls `deleteRec(root, 30)`
- In `deleteRec`:
  - $30 < 50$ , so we recurse to the left: `root.left = deleteRec(root.left, 30)`
  - We find 30. It has only a right child (40).
  - We return `root.right` (which is 40).
  - This 40 is assigned to the left child of 50.

After deletion, the tree looks like this:

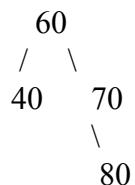


## 3. Deleting 50 (the root node with two children):

When we call `tree.deleteKey(50)`, here's what happens:

- `deleteKey(50)` calls `deleteRec(root, 50)`
- In `deleteRec`:
  - We find 50 (the root). It has two children.
  - We find the minimum value in the right subtree: `minValue(root.right)`
  - This returns 60.
  - We replace 50 with 60: `root.key = 60`
  - We delete 60 from the right subtree: `root.right = deleteRec(root.right, 60)`
  - This recursively deletes 60, which is a leaf node in the right subtree.

After deletion, the tree looks like this:



Here's a detailed look at the deleteRec method for the case of deleting 50:

```
java
Node deleteRec(Node root, int key) {
    // root is 50, key is 50
    if (root == null) return root;

    if (key < root.key)
        root.left = deleteRec(root.left, key);
    else if (key > root.key)
        root.right = deleteRec(root.right, key);
    else {
        // We're here because key == root.key (50 == 50)
        if (root.left == null)
            return root.right;
        else if (root.right == null)
            return root.left;

        // Node with two children: Get the inorder successor (smallest
        // in the right subtree)
        root.key = minValue(root.right); // This sets root.key to 60

        // Delete the inorder successor
        root.right = deleteRec(root.right, root.key);
        // This deletes 60 from the right subtree
    }

    return root;
}
```

The key aspects of the deletion process are:

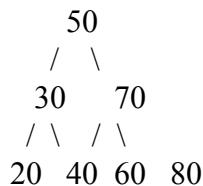
1. For leaf nodes, we simply return null, effectively removing them from the tree.
2. For nodes with one child, we return that child, effectively replacing the node with its child.
3. For nodes with two children, we replace the node's value with its inorder successor's value, then delete the inorder successor.

This process ensures that the BST property is maintained throughout the deletion operation.

---

### Detailed Dry Run of BST Deletion:

Let's start with a binary search tree containing 7 nodes:

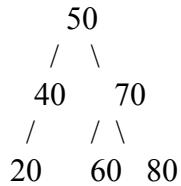


## Initial Deletion: Node 30

Let's delete node 30, which has two children.

1. We start at the root (50) and traverse left to 30.
2. We find that 30 is the node to delete and it has two children.
3. We need to find the inorder successor (smallest node in the right subtree of 30).
  - Start at the right child of 30, which is 40.
  - 40 has no left child, so it is the inorder successor.
4. Replace 30's data with 40's data.
5. Delete the original 40 node by setting 30's right child to 40's right child (which is null).

The tree now looks like this:



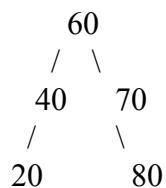
## Second Deletion: Node 50

Now let's delete the root node 50.

1. We start at the root (50) and find it's the node to delete.
2. 50 has two children, so we need to find its inorder successor.
3. We go to the right subtree (70) and then keep going left until we can't anymore.
  - Right child of 50 is 70.
  - Left child of 70 is 60.
  - 60 has no left child, so it's our inorder successor.

- 
- 4. Replace 50's data with 60's data.
  - 5. Delete the original 60 node:
    - Update 70's left child to point to 60's right child (which is null).

The tree now looks like this:

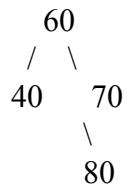


#### ## Third Deletion: Node 20

Let's delete a leaf node, 20.

- 1. We start at the root (60) and go left to 40, then left to 20.
- 2. We find 20 is the node to delete and it has no children.
- 3. We simply return null to its parent (40), effectively removing it from the tree.

The tree now looks like this:



#### ## Fourth Deletion: Node 70

Finally, let's delete a node with one child, 70.

- 1. We start at the root (60) and go right to 70.
- 2. We find 70 is the node to delete and it has one child (80).
- 3. We return 70's right child (80) to replace 70 in the tree.

The final tree looks like this:

60



This dry run demonstrates how the deleteRec function handles different scenarios:

1. Deleting a node with two children (30 and 50)
2. Deleting a leaf node (20)
3. Deleting a node with one child (70)

In each case, the binary search tree property is maintained, ensuring that for each node, all elements in its left subtree are smaller and all elements in its right subtree are larger.

# Day 13 - Dynamic Programming, Lambda and Practice

**Dynamic Programming** (DP) is an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems.

## Program 66 - Program to Reverse a String

The screenshot shows a Java code editor with the following code:

```
Java > Code > P66ReverseWord.java > P66ReverseWord
1 import java.util.Stack;
2
3 public class P66ReverseWord {
4
5     public static String reverseWords(String sentence) {
6         Stack<String> stack = new Stack<>();
7         String reversedSentence = "";
8         String[] words = sentence.split(regex:" ");
9
10        // Push each word onto the stack
11        for (String word : words) {
12            stack.push(word);
13        }
14
15        // Pop each word from the stack to form the reversed sentence
16        while (!stack.isEmpty()) {
17            reversedSentence += stack.pop();
18            if (!stack.isEmpty()) {
19                reversedSentence += " ";
20            }
21        }
22
23        return reversedSentence;
24    }
25
26    Run | Debug
27    public static void main(String[] args) {
28        String sentence = "hello world";
29        System.out.println("Original Sentence: " + sentence);
30        System.out.println("Reversed Sentence: " + reverseWords(sentence));
31    }
32
33 }
```

At the bottom of the code editor, there is a terminal window showing the output of running the program:

```
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Codebase> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Anjali\AppData\Roaming\redhat.java\jdt_ws\jdt.ls\java-project\bin' 'P66ReverseWord'
Original Sentence: hello world
Reversed Sentence: world hello
PS C:\Codebase>
```

## Program 67 - Dynamic programming solution for calculating the Fibonacci sequence

```
Java > Code > J P67FibonacciDP.java > P67FibonacciDP > main(String[])
1  public class P67FibonacciDP {
2
3      // Function to calculate Fibonacci using dynamic programming
4      public static int fibonacci(int n) {
5          // Create an array to store the Fibonacci numbers up to n
6          int[] dp = new int[n + 1];
7
8          // Base cases for Fibonacci sequence
9          dp[0] = 0; // Fibonacci(0) = 0
10         if (n > 0) {
11             dp[1] = 1; // Fibonacci(1) = 1
12         }
13
14         // Calculate Fibonacci numbers from 2 to n using the bottom-up approach
15         for (int i = 2; i <= n; i++) {
16             dp[i] = dp[i - 1] + dp[i - 2];
17         }
18
19         // Return the nth Fibonacci number
20         return dp[n];
21     }
22
23     // Correct main method signature
Run | Debug
24     public static void main(String[] args) {
25         int n = 10; // Example input, you can change it to any other value
26         System.out.println("Fibonacci of " + n + " is: " + fibonacci(n));
27     }
28 }
29
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Anjali\AppData\Local\Temp\workspaceStorage\dc86b2a86068cf587cb193dcf15e39aa\redhat.java\jdt_ws\jdt_ls-java-project\bin' 'FibonacciDP'
Fibonacci of 10 is: 55
PS C:\Codebase>
```

## Program 68 - Dynamic programming solution for the coin change problem

```
Java > Code > J CoinChangeDP.java > ...
1 import java.util.Arrays;
2
3 public class CoinChangeDP {
4
5     public static int coinChange(int[] coins, int amount) {
6         // Create an array dp[] of size amount + 1 and initialize it to a large value (infinity)
7         int[] dp = new int[amount + 1];
8         Arrays.fill(dp, amount + 1); // We fill with a large number greater than the maximum possible coins
9         dp[0] = 0; // Base case: to make amount 0, we need 0 coins
10
11        // Iterate through each amount from 1 to 'amount'
12        for (int i = 1; i <= amount; i++) {
13            // For each coin, update the dp array if the coin can be used to make the current amount 'i'
14            for (int coin : coins) {
15                if (coin <= i) {
16                    // Calculate the minimum coins required for the current amount
17                    dp[i] = Math.min(dp[i], dp[i - coin] + 1);
18                }
19            }
20        }
21
22        // If dp[amount] is still a large number, return -1 (amount cannot be formed by given coins)
23        return dp[amount] > amount ? -1 : dp[amount];
24    }
25
26    Run | Debug
27    public static void main(String[] args) {
28        int[] coins = {1, 2, 5}; // Example coins
29        int amount = 11; // Example amount
30        int result = coinChange(coins, amount);
31
32        if (result != -1) {
33            System.out.println("Minimum coins required: " + result);
34        } else {
35            System.out.println("It's not possible to form the amount with the given coins.");
36        }
37    }
38}
```

```
PROBLEMS 13 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Codebase> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Anjali\AppData\Roaming\Cursor\>User\workspaceStorage\dc86b2a86068cf587cb193dcf15e39aa\redhat.java\jdt_ws\jdt.ls-java-project\bin' 'CoinChangeDP'
Minimum coins required: 3
PS C:\Codebase>
```

Initialize a `dp` array of size `amount + 1` with infinity ( $\infty$ ), except  $dp[0] = 0$  (0 coins to make 0).

Iterate through each coin and for every amount `x`, update  $dp[x]$  to the minimum coins needed ( $dp[x] = \min(dp[x], dp[x - coin] + 1)$ ).

Final result is in `dp[amount]`. If it's still  $\infty$ , return `-1` (not possible to form the amount), otherwise return the minimum number of coins.

Amount: 0 1 2 3 4 5 6 7 8 9 10 11

`dp:` 0  $\infty$   $\infty$   $\infty$   $\infty$   $\infty$   $\infty$   $\infty$   $\infty$   $\infty$  (Initial)

`dp:` 0 1  $\infty$   $\infty$   $\infty$   $\infty$   $\infty$   $\infty$   $\infty$   $\infty$   $\infty$  (After 1)

`dp:` 0 1 1 2 2  $\infty$   $\infty$   $\infty$   $\infty$   $\infty$   $\infty$  (After 2)

`dp:` 0 1 1 2 2 1 2 2 3 3 2 3 (Final)

## Program 69 - An approach for checking anagrams

```
Java > Code > P69AnagramChecker.java > P69AnagramChecker
 1  public class P69AnagramChecker {
 2      public static boolean isAnagram(String str1, String str2) {
 3          // Remove all spaces and convert to lowercase
 4          str1 = str1.replaceAll(regex:"\\s", replacement:"").toLowerCase();
 5          str2 = str2.replaceAll(regex:"\\s", replacement:"").toLowerCase();
 6
 7          // If lengths differ, they can't be anagrams
 8          if (str1.length() != str2.length()) {
 9              return false;
10          }
11
12          // Array to count character frequencies
13          int[] freq = new int[26]; // Assuming only lowercase a-z characters
14
15          // Increment for characters in str1 and decrement for str2
16          for (int i = 0; i < str1.length(); i++) {
17              freq[str1.charAt(i) - 'a']++;
18              freq[str2.charAt(i) - 'a']--;
19          }
20
21          // Check if all frequencies are zero
22          for (int count : freq) {
23              if (count != 0) {
24                  return false;
25              }
26          }
27
28          return true; // If all counts are zero, it's an anagram
29      }
30
31      Run | Debug
32      public static void main(String[] args) {
33          String str1 = "Listen";
34          String str2 = "Silent";
35
36          if (isAnagram(str1, str2)) {
37              System.out.println(str1 + " and " + str2 + " are anagrams.");
38          } else {
39              System.out.println(str1 + " and " + str2 + " are not anagrams.");
40          }
41      }
42  }
```

PROBLEMS 14 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Anjali\AppData\orange\dc86b2a86068cf587cb193dcf15e39aa\redhat.java\jdt_ws\jdt.ls-java-project\bin' 'P69AnagramChecker'
Listen and Silent are anagrams.
PS C:\Codebase>
```

## Program 70 - Lambda Example

```
Java > Code > J LambdaExamples.java > ...
1 import java.util.Arrays;
2 import java.util.List;
3 import java.util.function.Function;
4 import java.util.function.Predicate;
5
6 public class LambdaExamples {
7     Run|Debug
8     public static void main(String[] args) {
9         // Basic Lambda syntax
10        Runnable runnable = () -> System.out.println("hello lambda!!!");
11        runnable.run();
12
13        // 2. Lambda with functional interface
14        Predicate<String> isLong = s -> s.length() > 5;
15        System.out.println(isLong.test("lambda"));
16
17        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
18        numbers.forEach(n -> System.out.println("Number: " + n));
19
20        Function<Integer, Integer> square = x -> x * x;
21        numbers.stream().map(square).forEach(System.out::println);
22
23        Function<Integer, Boolean> isEven = getIsEvenLambda();
24        System.out.println(isEven.apply(4));
25    }
26
27    // Function to check if a number is even
28    private static Function<Integer, Boolean> getIsEvenLambda() {
29        return t -> t % 2 == 0;
30    }
31}
```

PROBLEMS 15 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Anjali\AppData\Roaming\Cursor\User\orange\dc86b2a86068cf587cb193dcf15e39aa\redhat.java\jdt_ws\jdt.ls-java-project\bin' 'LambdaExamples'
hello lambda!!!
true
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
1
4
9
16
25
true
PS C:\Codebase>
```

# Day 14 - Strings, Double Hashing and Practice

## Program 71 - Expand-around-center approach to check for palindromes

The screenshot shows a Java code editor with the file `P71Solution.java` open. The code implements the expand-around-center approach to find the longest palindrome in a given string. It includes test cases for strings "babad", "cbbd", "a", and "ac". The output terminal at the bottom shows the execution results for each test case.

```
Java > Code > P71Solution.java > ...
1 class Solution {
2     Run|Debug
3     public static void main(String[] args) {
4         Solution solution = new Solution();
5
6         // Test cases
7         String test1 = "babad";
8         String test2 = "cbbd";
9         String test3 = "a";
10        String test4 = "ac";
11
12        // Running the method
13        System.out.println("Longest palindrome in 'babad': " + solution.longestPalindrome(test1));
14        System.out.println("Longest palindrome in 'cbbd': " + solution.longestPalindrome(test2));
15        System.out.println("Longest palindrome in 'a': " + solution.longestPalindrome(test3));
16        System.out.println("Longest palindrome in 'ac': " + solution.longestPalindrome(test4));
17    }
18
19    public String longestPalindrome(String s) {
20        if (s == null || s.length() < 2) {
21            return s;
22        }
23
24        int start = 0, maxLength = 1;
25
26        for (int i = 0; i < s.length(); i++) {
27            int len1 = expandAroundCenter(s, i, i);
28            int len2 = expandAroundCenter(s, i, i + 1);
29            int len = Math.max(len1, len2);
30
31            if (len > maxLength) {
32                start = i - (len - 1) / 2;
33                maxLength = len;
34            }
35        }
36
37        return s.substring(start, start + maxLength);
38    }
39
40    private int expandAroundCenter(String s, int left, int right) {
41        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
42            left--;
43            right++;
44        }
45        return right - left - 1;
46    }
}
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS
-java-project\bin' 'Solution'
Longest palindrome in 'babad': bab
Longest palindrome in 'cbbd': bb
Longest palindrome in 'cbbd': bb
Longest palindrome in 'a': a
Longest palindrome in 'ac': a
PS C:\Codebase>
```

---

## The time complexity of the algorithm represented using Big-O notation.

### 1. Graph of Time Complexities:

- There's a graph showing how different time complexities grow as input size increases. It seems to illustrate logarithmic growth ( $O(\log n)$ ) versus exponential growth.

### 2. Big-O Notation Hierarchy:

- $O(1)$  (constant time) is the fastest.
- $O(\log n)$  (logarithmic time) comes next.
- $O(n)$  (linear time).
- $O(n \log n)$  (linearithmic time).
- $O(n^2)$  (quadratic time).
- $O(2^n)$  (exponential time) is the slowest.

### 3. Logarithmic and Exponential Terms:

- The logarithmic equation  $\log_{10} = O(1)$  shows a base example where the result is constant.
- For exponential time  $O(2^n)$ , it points out that  $2^{10}$  is an example of how rapidly it grows.

### 4. Hierarchy of Complexities:

- It highlights the order of different time complexities:  
 $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n)$
- This is a common way to rank the efficiency of algorithms, with smaller Big-O values being faster.

## Program 72 - Binary Search Algorithm

```
Java > Code > J P72BinarySearch.java > P72BinarySearch > binarySearch(int[], int)
1  public class P72BinarySearch {
2      public static int binarySearch(int[] arr, int target) {
3          // Define the Left and right pointers
4          int left = 0, right = arr.length - 1;
5
6          // Loop until the two pointers meet
7          while (left <= right) {
8              // Calculate the middle index
9              int mid = (left + right) / 2;
10
11             // Check if the middle element is the target
12             if (arr[mid] == target) {
13                 return mid; // Target found at index mid
14             }
15
16             // If the target is greater, ignore the left half
17             if (arr[mid] < target) {
18                 left = mid + 1;
19             }
20
21             // If the target is smaller, ignore the right half
22             else {
23                 right = mid - 1;
24             }
25         }
26
27         // Return -1 if the target is not found
28         return -1;
29     }
30
31     Run|Debug
32     public static void main(String[] args) {
33         int[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
34         int target = 7;
35
36         int result = binarySearch(arr, target);
37         if (result != -1) {
38             System.out.println("Target found at index " + result);
39         } else {
40             System.out.println("Target not found");
41         }
42     }
43 }
```

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Codebase> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Anjali\AppData\Roaming\edhat.java\jdt_ws\jdt.ls\java-project\bin' 'P72BinarySearch'
Target found at index 6
PS C:\Codebase>
```

## Program 73 - Demonstration of Double Hashing

```
java > Code > P73App.java >
1  import java.security.KeyStore.Entry;
2
3  class HashTable<K,V>
4  {
5      private Entry<K,V>[] table;
6      private int size;
7      private static final double LOAD_FACTOR_THRESHOLD=0.75;
8
9      public HashTable(int capacity){
10         table = new Entry[capacity];
11         size = 0;
12     }
13
14     public void put(K key, V value){
15         if(key == null){ // Check if the key is null to prevent null pointer exceptions
16             throw new IllegalArgumentException("Null keys are not allowed"); // Throw an exception if the key is null
17         }
18         if((double)size / table.length >= LOAD_FACTOR_THRESHOLD){ // Check if the Load factor has exceeded the threshold
19             resize(); // Resize the table if the load factor has exceeded the threshold
20         }
21         int index = hash1(key); // Calculate the initial index using the first hash function
22         int step = hash2(key); // Calculate the step size using the second hash function
23         int i = 0; // Initialize the iteration counter
24         while(table[index] != null && table[index].getKey().equals(key)){ // Loop until an empty slot is found or the key is found
25             index = (index + step * i) % table.length; // Calculate the next index using the step size and iteration counter
26             i++; // Increment the iteration counter
27         }
28         table[index] = new Entry<K,V>(key, value); // Insert the new entry into the table
29         size++; // Increment the size of the table
30     }
31
32     public V get(K key){
33         if(key == null){ // Check if the key is null to prevent null pointer exceptions
34             throw new IllegalArgumentException("Null keys are not allowed"); // Throw an exception if the key is null
35         }
36         int index = hash1(key);
37         int step = hash2(key);
38         int i = 0;
39         while(table[index]!=null){
40             if(table[index].getKey().equals(key)){
41                 return table[index].getValue();
42             }
43             index = (index + step * i) % table.length;
44             i++;
45         }
46         return null;
47     }
48
49     private void resize(){
50         Entry<K,V> oldTable;
51         table= new Entry[oldTable.length * 2];
52         size = 0;
53         for (Entry<K,V> entry:oldTable){
54             if (entry !=null){
55                 put(entry.getKey(),entry.getValue());
56             }
57         }
58         private int hash1(K key){
59             return Math.abs(key.hashCode())%table.length;
60         }
61         private int hash2(K key){
62             return 1+ (Math.abs(key.hashCode()))%(table.length -1);
63         }
64         private static class Entry<K,V>{
65             private K key;
66             private V value;
67             public Entry(K key, V value){
68                 this.key = key;
69                 this.value = value;
70             }
71             public K getKey(){
72                 return key;
73             }
74             public V getValue(){
75                 return value;
76             }
77         }
78
79     }
80
81     /* Ctrl+L to chat, Ctrl+K to generate
82     */
83     /* Hello world!
84     */
85     */
86     public class P73App
87     {
88         Run|Debug
89         public static void main( String[] args )
90         {
91             System.out.println( "Hello World!" );
92         }
93     }
94
95     PROBLEMS 15 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Codebase> <C
PS C:\Codebase>
PS C:\Codebase> c:; cd 'c:\Codebase'; & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Anjali\AppData\cF587ch193dcf15e39aa\redhat.java\dtls-ws\jdt-ls-java-project\bin' 'P73App'
Hello World!
```

## Program 74 - A basic hash table data structure with double hashing for collision resolution

```
Java > Code > J P74App.java > ...
1  class HashTable<K, V> {
2      private Entry<K, V>[] table;
3      private int size;
4      private static final double LOAD_FACTOR_THRESHOLD = 0.75;
5
6      @SuppressWarnings("unchecked")
7      public HashTable(int capacity) {
8          table = (Entry<K, V>[] ) new Entry[capacity]; // Initialize the table array
9          size = 0;
10     }
11
12     public void put(K key, V value) {
13         if (key == null) {
14             throw new IllegalArgumentException("Null keys are not allowed");
15         }
16         if ((double) size / table.length >= LOAD_FACTOR_THRESHOLD) {
17             resize(); // Resize if Load factor exceeds threshold
18         }
19         int index = hash1(key);
20         int step = hash2(key);
21         int i = 0;
22
23         // Handle collisions using double hashing
24         while (table[index] != null && !table[index].getKey().equals(key)) {
25             index = (index + step * i) % table.length; // Compute new index
26             i++;
27         }
28
29         table[index] = new Entry<>(key, value); // Insert key-value pair in the table
30         size++;
31     }
32
33     public V get(K key) {
34         if (key == null) {
35             throw new IllegalArgumentException("Null keys are not allowed");
36         }
37
38         int index = hash1(key);
39         int step = hash2(key);
40         int i = 0;
41
42         // Traverse the table until the key is found or an empty slot is encountered
43         while (table[index] != null) {
44             if (table[index].getKey().equals(key)) { // Call getKey() method correctly
45                 return table[index].getValue(); // Return the value if the key is found
46             }
47             index = (index + step * i) % table.length; // Calculate the next index
48             i++;
49         }
50
51         return null; // Return null if the key is not found
52     }
53
54     @SuppressWarnings("unchecked")
55     private void resize() {
```

```

55     private void resize() {
56         Entry<K, V>[] oldTable = table; // Keep a reference to the old table
57         table = (Entry<K, V>[] ) new Entry[oldTable.length * 2]; // Create a new table with double capacity
58         size = 0;
59
60         // Rehash and reinsert all old entries into the new table
61         for (Entry<K, V> entry : oldTable) {
62             if (entry != null) {
63                 put(entry.getKey(), entry.getValue()); // Reinsert key-value pairs
64             }
65         }
66     }
67
68     private int hash1(K key) {
69         return Math.abs(key.hashCode()) % table.length;
70     }
71
72     private int hash2(K key) {
73         return 1 + (Math.abs(key.hashCode()) % (table.length - 1));
74     }
75
76     // Static inner class to represent an entry in the hash table
77     private static class Entry<K, V> {
78         private final K key;
79         private final V value;
80
81         public Entry(K key, V value) {
82             this.key = key;
83             this.value = value;
84         }
85
86         public K getKey() {
87             return key;
88         }
89
90         public V getValue() {
91             return value;
92         }
93     }
94 }
95
96 // Main application class
97 public class P74App {
98     Run|Debug
99     public static void main(String[] args) {
100         HashTable<String, Integer> hashTable = new HashTable<> (capacity:16); // Initialize hash table with capacity 16
101
102         // Adding key-value pairs to the hash table
103         hashTable.put(key:"apple", value:1);
104         hashTable.put(key:"banana", value:2);
105         hashTable.put(key:"orange", value:3);
106
107         // Retrieve and print value for key "banana"
108         System.out.println("Value for banana: " + hashTable.get(key:"banana")); // Output: 2
109
110         // Retrieve and print value for key "banana"
111         System.out.println("Value for banana: " + hashTable.get(key:"banana")); // Output: 2
112
113         // You can also test other key-value pairs
114         System.out.println("Value for apple: " + hashTable.get(key:"apple")); // Output: 1
115         System.out.println("Value for orange: " + hashTable.get(key:"orange")); // Output: 3
116         System.out.println("Value for mango: " + hashTable.get(key:"mango")); // Output: null (key not present)
117     }
118 }

```

```

106         // Retrieve and print value for key "banana"
107         System.out.println("Value for banana: " + hashTable.get(key:"banana")); // Output: 2
108
109         // You can also test other key-value pairs
110         System.out.println("Value for apple: " + hashTable.get(key:"apple")); // Output: 1
111         System.out.println("Value for orange: " + hashTable.get(key:"orange")); // Output: 3
112         System.out.println("Value for mango: " + hashTable.get(key:"mango")); // Output: null (key not present)
113     }
114 }
115 | Ctrl+L to chat, Ctrl+K to generate

```

## Concept of Leaders:

The concept of "leaders" in an array refers to elements that are greater than all the elements to their right. The rightmost element is always a leader because there are no elements to its right.

### Program 75 - Concept of Leaders

The screenshot shows a Java code editor with the following code:

```
Java > Code > J LeadersInArray.java > P75LeadersInArray > findLeaders(int[])
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class P75LeadersInArray {
5     Run | Debug
6     public static void main(String[] args) {
7         int[] arr1 = {4, 7, 1, 0};
8         int[] arr2 = {10, 22, 12, 3, 0, 6};
9
10        System.out.println("Leaders in arr1: " + findLeaders(arr1));
11        System.out.println("Leaders in arr2: " + findLeaders(arr2));
12    }
13
14    public static List<Integer> findLeaders(int[] arr) {
15        List<Integer> leaders = new ArrayList<>();
16        int n = arr.length;
17        int maxFromRight = arr[n - 1]; // Rightmost element is always a Leader
18        leaders.add(maxFromRight);
19
20        // Traverse the array from right to left
21        for (int i = n - 2; i >= 0; i--) {
22            if (arr[i] > maxFromRight) {
23                leaders.add(arr[i]); // Current element is a leader
24                maxFromRight = arr[i]; // Update maxFromRight
25            }
26        }
27
28        // Reverse the Leaders list to maintain the order
29        List<Integer> orderedLeaders = new ArrayList<>();
30        for (int i = leaders.size() - 1; i >= 0; i--) {
31            orderedLeaders.add(leaders.get(i));
32        }
33
34        return orderedLeaders;
35    }
36}
```

Below the code editor is a terminal window showing the output of the program:

```
PROBLEMS 14 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Codebase> c:; cd 'c:\Codebase' & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\redhat.java\jdt_ws\jdt.ls-java-project\bin' 'LeadersInArray'
Exception in thread "main" java.lang.Error: Unresolved compilation problem:

        at LeadersInArray.main(P75LeadersInArray.java:5)
PS C:\Codebase> ^C
PS C:\Codebase>
PS C:\Codebase> c:; cd 'c:\Codebase' & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\redhat.java\jdt_ws\jdt.ls-java-project\bin' 'LeadersInArray'
Leaders in arr1: [7, 1, 0]
Leaders in arr2: [22, 12, 6]
PS C:\Codebase> []
```

## Day 15 - SQL Commands

```
1 •  create database testdb;
2 •  show databases;
3 •  use testdb;
4 •  show tables;
5 •  CREATE TABLE key_value_table (
6      id int NOT NULL,
7      value varchar(255) DEFAULT NULL,
8      PRIMARY KEY (id)
9  );
10 • select * from key_value_table;
11 • alter table key_value_table rename to key_value_pair;
12 • select * from key_value_pair;
13 • insert into key_value_pair values(1,'asgdjhasd');
14 • show create table key_value_pair;
15 •  CREATE TABLE key_val_pair (
16      id int NOT NULL,
17      value varchar(255) DEFAULT NULL,
18      PRIMARY KEY (id)
19  );
20 • truncate table key_value_pair;
21 • select * from key_val_pair;
22 • CREATE TABLE key_val_pair AS
23     SELECT * FROM key_value_pair;
24 • INSERT INTO key_val_pair SELECT * FROM key_value_pair;
25 • select * from key_val_pair;
26
27 • insert into key_value_pair value(2,null);
28 • UPDATE key_value_pair SET VALUE ='Jaggu' WHERE VALUE IS NULL;
29 • select * from key_value_pair LIMIT 2;
```

```

30 •   SELECT MIN(ID) FROM key_value_pair;
31 •   SELECT MAX(ID) FROM key_value_pair;
32
33 •   select count(id) from key_value_pair;
34 •   select sum(id) from key_value_pair;
35
36 •   select avg(id) from key_value_pair;
37
38 •   select * from key_value_pair where value like '%j%';
39 •   select * from key_value_pair where value like '__g%';
40
41 •   select * from key_value_pair where value in('asgdjhasd');
42 •   select * from key_value_pair where id between 1 and 100;
43
44 •   select * from key_value_pair as table_value;

```

	id	value
▶	1	asgdjhasd
▶	2	Jaggu
*	NULL	NULL

The brief explanation of the above commands used:

### 1. Database and Table Management:

- CREATE DATABASE testdb;: Creates a new database called `testdb`.
- SHOW DATABASES;: Lists all available databases.
- USE testdb;: Switches the session to use the `testdb` database.
- SHOW TABLES;: Lists all the tables in the currently selected database.
- CREATE TABLE key\_value\_table (...);: Creates a new table `key_value_table` with columns `id` (integer, primary key) and `value` (optional string).
- ALTER TABLE key\_value\_table RENAME TO key\_value\_pair;: Renames the table `key_value_table` to `key_value_pair`.

- `INSERT INTO key_value_pair VALUES (1, 'asgdjhasd');`: Inserts a row into the table with `id=1` and `value='asgdjhasd'`.
- `SHOW CREATE TABLE key_value_pair;`: Displays the `CREATE TABLE` statement of `key_value_pair`.

## 2. Table Duplication:

- `CREATE TABLE key_val_pair AS SELECT * FROM key_value_pair;`: Creates a new table `key_val_pair` with the same structure and data as `key_value_pair`.
- `INSERT INTO key_val_pair SELECT * FROM key_value_pair;`: Copies the data from `key_value_pair` to `key_val_pair`.

## 3. Updating and Deleting Data:

- `INSERT INTO key_value_pair VALUES (2, NULL);`: Inserts a row with `id=2` and `value=NULL`.
- `UPDATE key_value_pair SET VALUE ='Jaggu' WHERE VALUE IS NULL;`: Updates all rows where `value` is `NULL`, setting `value` to '`Jaggu`'.
- `TRUNCATE TABLE key_value_pair;`: Deletes all data from the table `key_value_pair` but keeps its structure.

## 4. Selection and Filtering Data:

- `SELECT * FROM key_value_pair;`: Retrieves all rows and columns from the `key_value_pair` table.
- `SELECT * FROM key_value_pair LIMIT 2;`: Retrieves only the first 2 rows.
- `SELECT MIN(ID), MAX(ID) FROM key_value_pair;`: Retrieves the minimum and maximum `id` values.
- `SELECT COUNT(id)`: Counts the number of non-null `id` values.
- `SELECT SUM(id)`: Calculates the sum of all `id` values.
- `SELECT AVG(id)`: Calculates the average of `id` values.
- `SELECT * FROM key_value_pair WHERE value LIKE '%j%';`: Selects rows where `value` contains the letter 'j'.

- `SELECT * FROM key_value_pair WHERE value LIKE '__g%';`: Selects rows where value has 'g' as the third character.
- `SELECT * FROM key_value_pair WHERE value IN ('asgdjhasd');`: Selects rows where value matches 'asgdjhasd'.
- `SELECT * FROM key_value_pair WHERE id BETWEEN 1 AND 100;`: Selects rows where id is between 1 and 100.
- `SELECT * FROM key_value_pair AS table_value;`: Selects all rows from key\_value\_pair and gives it an alias table\_value.

This covers basic database management, data insertion, selection, filtering, updating, and understanding the structure of the database and tables.

### Some more SQL Commands:

```

50    -- Create tables
51 • ⏷ CREATE TABLE departments (
52     department_id INT PRIMARY KEY,
53     department_name VARCHAR(50),
54     location_id INT
55 );
56
57 • ⏷ CREATE TABLE locations (
58     location_id INT PRIMARY KEY,
59     city VARCHAR(50)
60 );
61
62 • ⏷ CREATE TABLE employees (
63     employee_id INT PRIMARY KEY,
64     first_name VARCHAR(50),
65     last_name VARCHAR(50),
66     email VARCHAR(100),
67     phone_number VARCHAR(20),
68     hire_date DATE,
69     job_id VARCHAR(10),
70     salary DECIMAL(10, 2),
71     department_id INT,
72     FOREIGN KEY (department_id) REFERENCES departments(department_id)
73 );

```

```
75 • Ⓜ CREATE TABLE customers (
76     customer_id INT PRIMARY KEY,
77     customer_name VARCHAR(100),
78     email VARCHAR(100)
79 );
80
81 • Ⓜ CREATE TABLE orders (
82     order_id INT PRIMARY KEY,
83     customer_id INT,
84     order_date DATE,
85     FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
86 );
87
88 • Ⓜ CREATE TABLE products (
89     product_id INT PRIMARY KEY,
90     product_name VARCHAR(100),
91     category VARCHAR(50),
92     price DECIMAL(10, 2)
93 );
94
95 • Ⓜ CREATE TABLE discontinued_products (
96     product_id INT PRIMARY KEY,
97     product_name VARCHAR(100),
98     category VARCHAR(50)
99 );
```

```

101  -- Insert sample data
102 • INSERT INTO departments (department_id, department_name, location_id) VALUES
103  (10, 'Administration', 1),
104  (20, 'Marketing', 2),
105  (30, 'Purchasing', 1),
106  (40, 'Human Resources', 2),
107  (50, 'Shipping', 3);
108
109 • INSERT INTO locations (location_id, city) VALUES
110  (1, 'New York'),
111  (2, 'Los Angeles'),
112  (3, 'Chicago');
113
114 • INSERT INTO employees (employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary, department_id) VALUES
115  (1, 'John', 'Doe', 'john.doe@example.com', '515.123.4567', '2019-06-17', 'AD_PRES', 24000.00, 10),
116  (2, 'Jane', 'Smith', 'jane.smith@example.com', '515.123.4568', '2020-02-20', 'AD_VP', 17000.00, 10),
117  (3, 'Alice', 'Johnson', 'alice.johnson@example.com', '515.123.4569', '2020-08-11', 'MK_MAN', 9000.00, 20),
118  (4, 'Bob', 'Brown', 'bob.brown@example.com', '515.123.4560', '2021-03-05', 'HR REP', 6000.00, 40),
119  (5, 'Charlie', 'Davis', 'charlie.davis@example.com', '515.123.4561', '2021-11-30', 'SH_CLERK', 3000.00, 50);
120
121 • INSERT INTO customers (customer_id, customer_name, email) VALUES
122  (1, 'Acme Corp', 'contact@acmecorp.com'),
123  (2, 'GlobalTech', 'info@globaltech.com'),
124  (3, 'Local Shop', 'owner@localshop.com');

126 • INSERT INTO orders (order_id, customer_id, order_date) VALUES
127  (1, 1, '2023-01-15'),
128  (2, 1, '2023-02-20'),
129  (3, 2, '2023-02-22');
130
131 • INSERT INTO products (product_id, product_name, category, price) VALUES
132  (1, 'Laptop', 'Electronics', 999.99),
133  (2, 'Smartphone', 'Electronics', 699.99),
134  (3, 'Desk Chair', 'Furniture', 199.99);
135
136 • INSERT INTO discontinued_products (product_id, product_name, category) VALUES
137  (101, 'Old Laptop Model', 'Electronics'),
138  (102, 'Discontinued Phone', 'Electronics');

```

```

142      -- Queries
143
144 •  select e.first_name, e.last_name, d.department_name
145   from employees e
146   inner join departments d
147   on e.department_id= d.department_id;
148
...

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	first_name	last_name	department_name
▶	John	Doe	Administration
	Jane	Smith	Administration
	Alice	Johnson	Marketing
	Bob	Brown	Human Resources
	Charlie	Davis	Shipping

An INNER JOIN in SQL is used to combine rows from two or more tables based on a related column. The result of an INNER JOIN will only include rows where the join condition is met (i.e., the values in the specified columns match between the tables).

In this query:

- The `employees` table (aliased as `e`) is joined with the `departments` table (aliased as `d`) using the INNER JOIN.
- The condition for joining is that `e.department_id` (from the `employees` table) must equal `d.department_id` (from the `departments` table).
- The query retrieves:
  - `first_name` and `last_name` from the `employees` table.
  - `department_name` from the `departments` table.

The result shows employees' names and the name of the department they belong to, but only if they have a matching `department_id` in both tables.

```

150 •   select c.customer_name , o.order_id
151     from customers c
152     left join orders o on c.customer_id =o.customer_id;
153
154
155

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	customer_name	order_id
▶	Acme Corp	1
	Acme Corp	2
	GlobalTech	3
	Local Shop	NULL

A LEFT JOIN (or LEFT OUTER JOIN) in SQL returns all rows from the left table (the table mentioned first), and the matching rows from the right table. If there is no match, the result will contain **NULL** values for the columns from the right table.

SELECT:

- `c.customer_name`: Retrieves the `customer_name` from the `customers` table.
- `o.order_id`: Retrieves the `order_id` from the `orders` table.

FROM `customers c`:

- The query starts with the `customers` table, assigning it the alias `c`.

LEFT JOIN `orders o`:

- The `customers` table is left-joined with the `orders` table (aliased as `o`).
- The join condition is that the `customer_id` column from `customers` must match the `customer_id` column from `orders`.

The result grid shows customer names from the `customers` table and their associated `order_id` from the `orders` table.

If a customer has no matching order (like "Local Shop"), the `order_id` will be **NULL**.

```
156 •  select c.customer_name , o.order_id  
157   from customers c  
158   right join orders o on c.customer_id =o.customer_id;  
159  
160
```

Result Grid		
	customer_name	order_id
▶	Acme Corp	1
	Acme Corp	2
	GlobalTech	3

A RIGHT JOIN (or RIGHT OUTER JOIN) is similar to a LEFT JOIN, but it includes all rows from the right table and matches from the left table. If there is no match, `NULL` values will be displayed for the columns from the left table.

1. SELECT:
  - `c.customer_name`: Retrieves the `customer_name` from the `customers` table.
  - `o.order_id`: Retrieves the `order_id` from the `orders` table.
2. FROM `customers` `c`:
  - The query starts with the `customers` table, aliasing it as `c`.
3. RIGHT JOIN `orders` `o`:
  - The `customers` table is right-joined with the `orders` table (alias `o`).
  - The join condition is that the `customer_id` column from `customers` must match the `customer_id` column from `orders`.

The result grid shows the customer names from the `customers` table and the corresponding `order_id` from the `orders` table.

```
161 • select first_name, last_name, salary  
162   from employees  
163   where salary >(select avg(salary) from employees);  
164
```

Result Grid			
	first_name	last_name	salary
▶	John	Doe	24000.00
	Jane	Smith	17000.00

This query retrieves the details of employees who have a salary greater than the average salary of all employees.

1. SELECT first\_name, last\_name, salary:
  - Retrieves the `first_name`, `last_name`, and `salary` columns from the `employees` table.
2. FROM employees:
  - Specifies that the data is being pulled from the `employees` table.
3. WHERE salary > (SELECT avg(salary) FROM employees):
  - The WHERE clause filters the rows where the `salary` is greater than the result of the subquery.
  - The subquery (`SELECT avg(salary) FROM employees`) calculates the average salary of all employees.
  - Only employees with a salary higher than this average will be selected.

```
165 • select department_id , avg(salary) as avgsalary  
166   from employees  
167   group by department_id  
168   having avgsalary >10000;  
...
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	department_id	avgsalary		
▶	10	20500.000000		

This query retrieves the average salary of employees for each department but only includes departments where the average salary exceeds 10,000.

1. SELECT department\_id, AVG(salary) AS avgsalary:
  - This part selects the `department_id` and calculates the average salary using the `AVG()` function.
  - The average salary is given an alias `avgsalary` for easier reference in the output.
2. FROM employees:
  - Specifies that the data is being pulled from the `employees` table.
3. GROUP BY department\_id:
  - This groups the results by `department_id`, meaning that the average salary calculation will be performed for each department separately.
4. HAVING avgsalary > 10000:
  - The HAVING clause filters the results of the grouped query.
  - It ensures that only those groups (departments) with an average salary greater than 10,000 are included in the final result.

```

170 • select
171   d.department_name,
172   e.first_name,
173   e.last_name,
174   e.salary,
175   (select avg(salary) from employees where department_id = e.department_id) as dept_avg_salary
176   from
177   employees e
178   inner join
179   departments d
180   on e.department_id = d.department_id
181   where e.salary > (select avg(salary) from employees )
182   order by d.department_name , e.salary desc;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

department_name	first_name	last_name	salary	dept_avg_salary
Administration	John	Doe	24000.00	20500.000000
Administration	Jane	Smith	17000.00	20500.000000

This query retrieves details about employees whose salaries are above the overall average salary, along with their department names and the average salary of their respective departments.

#### 1. SELECT:

- `d.department_name`: Retrieves the name of the department from the `departments` table.
- `e.first_name, e.last_name, e.salary`: Retrieves the employee's first name, last name, and salary from the `employees` table.
- The subquery `(SELECT avg(salary) FROM employees WHERE department_id = e.department_id)` calculates the average salary of the department to which each employee belongs, giving it an alias `dept_avg_salary`.

#### 2. FROM employees e:

- Specifies the primary table, `employees`, with the alias `e`.

#### 3. INNER JOIN departments d:

- Joins the `departments` table (aliased as `d`) to the `employees` table based on the common `department_id`.

#### 4. ON e.department\_id = d.department\_id:

- This is the condition for the join, matching the `department_id` in both tables.

#### 5. WHERE e.salary > (SELECT avg(salary) FROM employees):

- Filters the results to include only those employees whose salary is greater than the overall average salary calculated by the subquery.

6. ORDER BY d.department\_name, e.salary DESC:
- Sorts the final result first by the `department_name` in ascending order, and then by `salary` in descending order within each department.

```

186 •  select c.customer_name,
187    count(o.order_id) as order_count,
188    coalesce(sum(p.price),0) as total_order_value
189  from
190  customers c
191  left join
192  orders o on c.customer_id = o.customer_id
193  left join
194  products p on p.product_id = (select product_id from orders where order_id = o.order_id limit 1)
195  group by
196  c.customer_id,c.customer_name
197  having
198  count(o.order_id)> 0
199  order by
200  total_order_value desc;

```

Result Grid			
	customer_name	order_count	total_order_value
▶	Acme Corp	6	3799.94
	GlobalTech	3	1899.97

This query retrieves the names of customers, the count of their orders, and the total value of those orders, while ensuring that only customers with at least one order are included in the results.

## 1. SELECT:

- `c.customer_name`: Retrieves the name of the customer from the `customers` table.
- `COUNT(o.order_id) AS order_count`: Counts the number of orders associated with each customer. This count is given the alias `order_count`.
- `COALESCE(SUM(p.price), 0) AS total_order_value`: Calculates the total value of all products ordered by the customer.
  - If there are no products (i.e., the sum is `NULL`), it returns `0` instead of `NULL`. This result is aliased as `total_order_value`.

## 2. FROM customers c:

- Specifies that the main table is `customers`, with an alias `c`.

3. **LEFT JOIN** `orders o ON c.customer_id = o.customer_id`:

  - Performs a **LEFT JOIN** on the `orders` table (aliased as `o`), matching the `customer_id` from the `customers` table to the `customer_id` in the `orders` table.
  - This ensures all customers are included, even if they have no orders.
4. **LEFT JOIN** `products p ON p.product_id = (SELECT product_id FROM orders WHERE order_id = o.order_id LIMIT 1)`:

  - This joins the `products` table (aliased as `p`) based on the product ID linked to the order.
  - The subquery retrieves a product ID for each order. The use of `LIMIT 1` ensures that only one product is selected per order.
  - Note that this subquery might not be efficient if there are multiple products per order.
5. **GROUP BY** `c.customer_id, c.customer_name`:

  - Groups the results by `customer_id` and `customer_name`, which is necessary for the aggregate functions (COUNT and SUM) to work correctly.
6. **HAVING COUNT(o.order\_id) > 0**:

  - Filters the results to include only customers who have placed at least one order.
7. **ORDER BY** `total_order_value DESC`:

  - Sorts the final results in descending order based on `total_order_value`, showing the highest total value first.

```

203 • select e1.first_name as employee1_first_name,
204   e1.last_name as employee1_last_name,
205   e2.first_name as employee2_first_name,
206   e2.last_name as employee2_last_name,
207   e1.job_id
208   from
209   employees e1
210   inner join
211   employees e2
212   on e1.job_id =e2.job_id and e1.employee_id<e2.employee_id
213

```

Result Grid				
	employee1_first_name	employee1_last_name	employee2_first_name	employee2_last_name
				job_id

This query retrieves pairs of employees who share the same job title. It uses a self-join on the `employees` table to achieve this.

### 1. SELECT:

- `e1.first_name AS employee1_first_name`: Retrieves the first name of the first employee (aliased as `employee1_first_name`).
- `e1.last_name AS employee1_last_name`: Retrieves the last name of the first employee (aliased as `employee1_last_name`).
- `e2.first_name AS employee2_first_name`: Retrieves the first name of the second employee (aliased as `employee2_first_name`).
- `e2.last_name AS employee2_last_name`: Retrieves the last name of the second employee (aliased as `employee2_last_name`).
- `e1.job_id`: Retrieves the job ID that both employees share.

### 2. FROM `employees e1`:

- Specifies the primary table, `employees`, and aliases it as `e1` for the first employee in the pair.

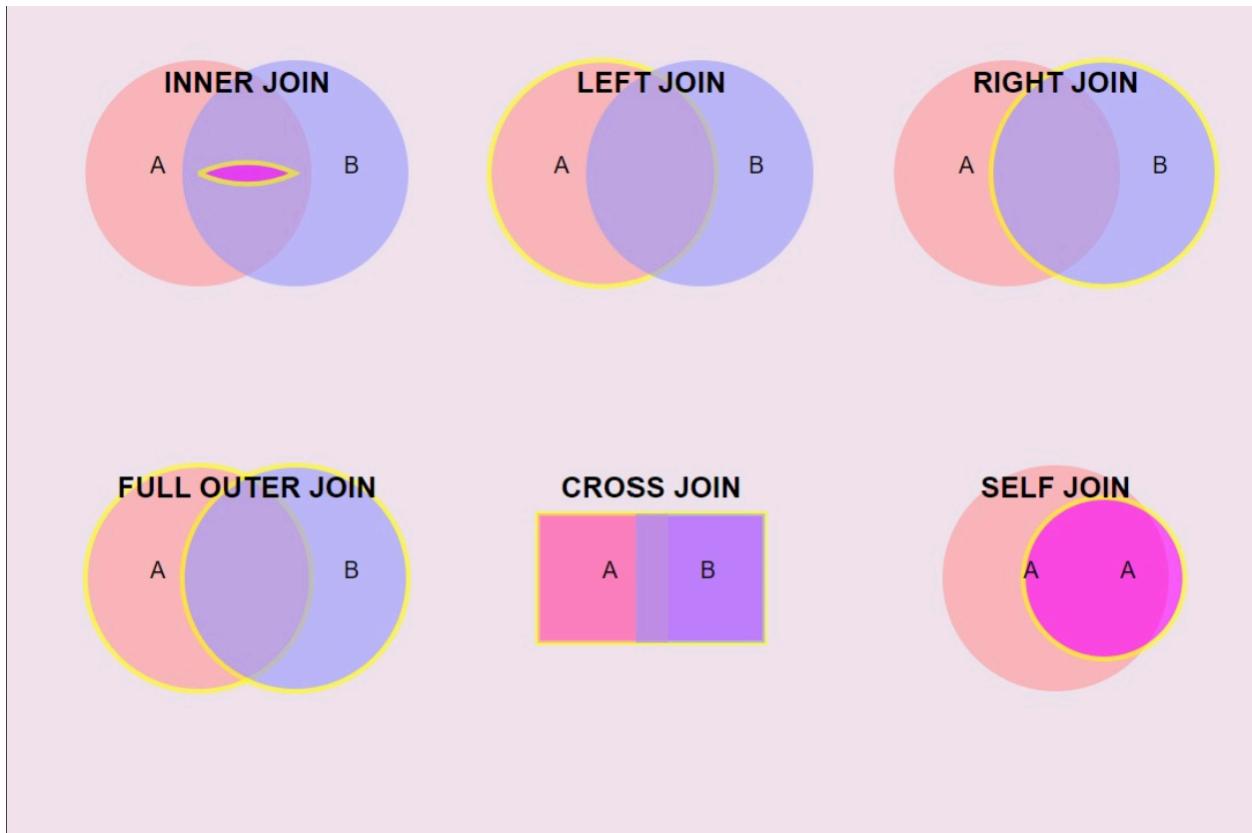
### 3. INNER JOIN `employees e2`:

- Joins the `employees` table again (aliased as `e2`) to match pairs of employees.

### 4. ON `e1.job_id = e2.job_id AND e1.employee_id < e2.employee_id`:

- This condition ensures that both employees share the same `job_id` (job title).

- The condition `e1.employee_id < e2.employee_id` ensures that each pair is unique and avoids duplicating pairs in reverse order (e.g., if John and Jane are paired, they won't appear again as Jane and John).



```

214 •  SELECT d.department_name,
215      e.first_name,
216      e.last_name,
217      e.salary,
218      AVG(e.salary) OVER (PARTITION BY d.department_id) AS dept_avg_salary,
219      RANK() OVER (PARTITION BY d.department_id ORDER BY e.salary DESC) AS salary_rank
220  FROM departments d
221  LEFT OUTER JOIN employees e ON d.department_id = e.department_id
222  ORDER BY d.department_name, e.salary DESC;
223
224
225

```

Result Grid | Filter Rows:  Export: Wrap Cell Content:

	department_name	first_name	last_name	salary	dept_avg_salary	salary_rank
▶	Administration	John	Doe	24000.00	20500.000000	1
	Administration	Jane	Smith	17000.00	20500.000000	2
	Human Resources	Bob	Brown	6000.00	6000.000000	1
	Marketing	Alice	Johnson	9000.00	9000.000000	1
	Purchasing	NULL	NULL	NULL	NULL	1
	Shipping	Charlie	Davis	3000.00	3000.000000	1

#### 1. **SELECT:**

- Retrieves the department name, employee's first name and last name, their salary, the average salary in their department, and their rank based on salary within the department.

#### 2. **AVG(e.salary) OVER (PARTITION BY d.department\_id):**

- Calculates the average salary for employees within the same department using the window function.

#### 3. **RANK() OVER (PARTITION BY d.department\_id ORDER BY e.salary DESC):**

- Assigns a rank to each employee based on their salary within their department, with the highest salary getting the highest rank.

#### 4. **LEFT OUTER JOIN:**

- Joins the `departments` table with the `employees` table, ensuring that all departments are included, even if they have no employees.

#### 5. **ORDER BY d.department\_name, e.salary DESC:**

- Orders the final result by department name and employee salary in descending order.

```

225 • select c.customer_name,
226   o.order_date,
227   case
228     when DAYOFWEEK(o.order_date) in (1,7) Then 'Weekend'
229     else 'Weekday'
230   end as order_day_type,
231   p.product_name,
232   p.price,
233   case
234     when p.price<100 then 'Budget'
235     when p.price between 100 and 500 then 'mid range'
236     else 'premium'
237   end as price_category
238   from customers c
239   inner join
240     orders o on c.customer_id =o.customer_id
241   inner join products p on p.product_id in (select product_id from orders where order_id =o.order_id)
242   where
243     o.order_date <= DATE_SUB(CURDATE(),INTERVAL 1 YEAR)
244   order by
245     o.order_date desc;

```

	customer_name	order_date	order_day_type	product_name	price	price_category
▶	GlobalTech	2023-02-22	Weekday	Laptop	999.99	premium
	GlobalTech	2023-02-22	Weekday	Smartphone	699.99	premium
	GlobalTech	2023-02-22	Weekday	Desk Chair	199.99	mid range
	Acme Corp	2023-02-20	Weekday	Laptop	999.99	premium
	Acme Corp	2023-02-20	Weekday	Smartphone	699.99	premium
	Acme Corp	2023-02-20	Weekday	Desk Chair	199.99	mid range
	Acme Corp	2023-01-15	Weekend	Laptop	999.99	premium
	Acme Corp	2023-01-15	Weekend	Smartphone	699.99	premium
	Acme Corp	2023-01-15	Weekend	Desk Chair	199.99	mid range

## SELECT:

- Retrieves customer names, order dates, product names, and product prices.
- Uses a **CASE** statement to determine if the order was placed on a weekend (**DAYOFWEEK(o.order\_date)** returns 1 for Sunday and 7 for Saturday) or a weekday.
- Another **CASE** statement categorizes the product based on its price into "Budget", "Mid Range", or "Premium".

---

## INNER JOINS:

- Joins the `customers` table to the `orders` table on `customer_id`.
- Joins the `products` table on `product_id` (directly using a foreign key relation).

## WHERE:

- Filters for orders that were placed at least one year ago (i.e., orders with `order_date` less than or equal to 1 year before the current date).

## ORDER BY:

- Orders the results by `order_date` in descending order (most recent orders first).

```
248 •  SELECT
249      c.customer_name,
250      o.order_date,
251      CASE
252          WHEN DAYOFWEEK(o.order_date) IN (1, 7) THEN 'Weekend'
253          ELSE 'Weekday'
254      END AS order_day_type,
255      p.product_name,
256      p.price,
257      CASE
258          WHEN p.price < 100 THEN 'Budget'
259          WHEN p.price BETWEEN 100 AND 500 THEN 'Mid Range'
260          ELSE 'Premium'
261      END AS price_category
262      FROM customers c
263      INNER JOIN orders o
264          ON c.customer_id = o.customer_id
265      INNER JOIN products p
266          ON p.product_id IN (1, 2, 3) -- Replace this with the actual relation of orders to products
267      WHERE o.order_date <= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
268      ORDER BY o.order_date DESC;
269
```

	customer_name	order_date	order_day_type	product_name	price	price_category
▶	GlobalTech	2023-02-22	Weekday	Laptop	999.99	Premium
	GlobalTech	2023-02-22	Weekday	Smartphone	699.99	Premium
	GlobalTech	2023-02-22	Weekday	Desk Chair	199.99	Mid Range
	Acme Corp	2023-02-20	Weekday	Laptop	999.99	Premium
	Acme Corp	2023-02-20	Weekday	Smartphone	699.99	Premium
	Acme Corp	2023-02-20	Weekday	Desk Chair	199.99	Mid Range
	Acme Corp	2023-01-15	Weekend	Laptop	999.99	Premium
	Acme Corp	2023-01-15	Weekend	Smartphone	699.99	Premium
	Acme Corp	2023-01-15	Weekend	Desk Chair	199.99	Mid Range

### Columns Selected:

- `c.customer_name`: The customer's name from the `customers` table.
- `o.order_date`: The date when the order was placed from the `orders` table.
- **Order Day Type**: The `CASE` statement checks whether the order was placed on a `Weekend` (if the day is Sunday (1) or Saturday (7)) or a `Weekday` (other days).
- `p.product_name`: The name of the product from the `products` table.
- `p.price`: The price of the product from the `products` table.
- **Price Category**: A second `CASE` statement assigns a category based on product price:
  - "Budget" for prices under 100,
  - "Mid Range" for prices between 100 and 500,
  - "Premium" for prices over 500.

### Tables Involved:

- **customers (c)**: The customers placing the orders.
- **orders (o)**: The orders made by the customers.
- **products (p)**: The products being ordered.

### Joins:

- **INNER JOIN orders o ON c.customer\_id = o.customer\_id**: Joins the `customers` table with the `orders` table on `customer_id`, linking each customer to their orders.

- 
- **INNER JOIN products p ON p.product\_id IN (1, 2, 3)**: Links the `products` table by matching product IDs (for simplicity, this is currently hard coded as 1, 2, 3; you should adjust this based on the actual relationship between `orders` and `products`).

**Filter:**

- **WHERE o.order\_date <= DATE\_SUB(CURDATE(), INTERVAL 1 YEAR)**: Filters orders to only include those placed within the last year.

**Order:**

- The results are ordered by `o.order_date` in descending order (**ORDER BY o.order\_date DESC**), so the most recent orders appear first.

## Day 16 - SQL Concepts

```
1      -- Example 2: Product Recommendations
2 •  create database product;
3 •  use product;
4 •  CREATE TABLE products (
5      product_id INT PRIMARY KEY,
6      product_name VARCHAR(100),
7      category VARCHAR(50),
8      price DECIMAL(10, 2)
9 );
10
11 •  CREATE TABLE product_recommendations (
12     product_id INT,
13     recommended_product_id INT,
14     strength DECIMAL(3, 2),
15     PRIMARY KEY (product_id, recommended_product_id),
16     FOREIGN KEY (product_id) REFERENCES products(product_id),
17     FOREIGN KEY (recommended_product_id) REFERENCES products(product_id)
18 );
19
20 •  INSERT INTO products (product_id, product_name, category, price) VALUES
21     (1, 'Laptop', 'Electronics', 999.99),
22     (2, 'Smartphone', 'Electronics', 699.99),
23     (3, 'Tablet', 'Electronics', 399.99),
24     (4, 'Headphones', 'Electronics', 149.99),
25     (5, 'Smart Watch', 'Electronics', 249.99);
26
27 •  INSERT INTO product_recommendations (product_id, recommended_product_id, strength) VALUES
```

```

27 •  INSERT INTO product_recommendations (product_id, recommended_product_id, strength) VALUES
28      (1, 2, 0.8),
29      (1, 3, 0.6),
30      (1, 4, 0.7),
31      (2, 1, 0.8),
32      (2, 3, 0.9),
33      (2, 5, 0.7),
34      (3, 1, 0.6),
35      (3, 2, 0.9),
36      (3, 4, 0.5);
37
38 •  select p1.product_name,
39       p2.product_name,
40       pr.strength
41   from product_recommendations pr
42   join products p1 on p1.product_id = pr.product_id
43   join products p2 on pr.recommended_product_id = p2.product_id
44   where p1.product_id = 1
45   order by pr.strength desc

```

	product_name	product_name	strength
▶	Laptop	Smartphone	0.80
	Laptop	Headphones	0.70
	Laptop	Tablet	0.60

Selects product names and the recommendation strength for a specific product (with `product_id = 1` in this case), showing which products are recommended based on the specified strength. The results are ordered by the `strength` in descending order.

This is used to retrieve product recommendations based on the `product_id`.

```
50      -- Example 3: Flight Connections
51 • CREATE TABLE airports (
52     airport_code CHAR(3) PRIMARY KEY,
53     airport_name VARCHAR(100),
54     city VARCHAR(50),
55     country VARCHAR(50)
56 );
57
58 • CREATE TABLE flights (
59     flight_id INT PRIMARY KEY,
60     departure_airport CHAR(3),
61     arrival_airport CHAR(3),
62     departure_time TIME,
63     arrival_time TIME,
64     FOREIGN KEY (departure_airport) REFERENCES airports(airport_code),
65     FOREIGN KEY (arrival_airport) REFERENCES airports(airport_code)
66 );
67
68 • INSERT INTO airports (airport_code, airport_name, city, country) VALUES
69     ('JFK', 'John F. Kennedy International Airport', 'New York', 'USA'),
70     ('LAX', 'Los Angeles International Airport', 'Los Angeles', 'USA'),
71     ('LHR', 'London Heathrow Airport', 'London', 'UK'),
72     ('CDG', 'Charles de Gaulle Airport', 'Paris', 'France'),
73     ('NRT', 'Narita International Airport', 'Tokyo', 'Japan');
74
75 • INSERT INTO flights (flight_id, departure_airport, arrival_airport, departure_time, arrival_time) VALUES
76     (1, 'JFK', 'LAX', '08:00', '11:00'),
77     (2, 'LAX', 'NRT', '13:00', '17:00'),
78     (3, 'NRT', 'LHR', '19:00', '23:00'),
```

```

--  

84    -- Find all possible one-stop flights from JFK to NRT  

85  

86 • select  

87     f1.flight_id,  

88     a1.city as departure_city,  

89     a2.city as layover_city,  

90     a3.city as arrival_city,  

91     f1.departure_time,  

92     f1.arrival_time as layover_arrival,  

93     f2.departure_time as layover_departure,  

94     f2.arrival_time  

95     from flights f1  

96     join flights f2 on f1.arrival_airport = f2.departure_airport  

97     join airports a1 on f1.departure_airport = a1.airport_code  

98     join airports a2 on f1.arrival_airport = a2.airport_code  

99     join airports a3 on f2.arrival_airport = a3.airport_code  

100    where f1.departure_airport ='JFK'  

101    and f2.arrival_airport ='NRT'  

102    and f2.departure_time> f1.arrival_time;  

103

```

Result Grid | Filter Rows:  Export: Wrap Cell Content:

	flight_id	departure_city	layover_city	arrival_city	departure_time	layover_arrival	layover_departure	arrival_time
▶	1	New York	Los Angeles	Tokyo	08:00:00	11:00:00	13:00:00	17:00:00

```

106 • CREATE TABLE employees (  

107     employee_id int PRIMARY KEY,  

108     first_name VARCHAR(100),  

109     last_name VARCHAR(50),  

110     manager_id int  

111 );  

112  

113 • INSERT INTO employees (employee_id, first_name, last_name, manager_id) VALUES  

114     (1, 'John', 'Doe', NULL),    -- CEO  

115     (2, 'Jane', 'Smith', 1),    -- Reports to John  

116     (3, 'Bob', 'Johnson', 1),   -- Reports to John  

117     (4, 'Alice', 'Williams', 2), -- Reports to Jane  

118     (5, 'Charlie', 'Brown', 2), -- Reports to Jane  

119     (6, 'David', 'Lee', 3),     -- Reports to Bob  

120     (7, 'Eva', 'Garcia', 3),    -- Reports to Bob  

121     (8, 'Frank', 'Lopez', 4),   -- Reports to Alice  

122     (9, 'Grace', 'Kim', 6),     -- Reports to David  

123     (10, 'Henry', 'Chen', 7);

```

```
125 • with recursive employee_hierarchy as (
126     select employee_id, first_name, last_name , manager_id,0 as level
127     from employees
128     where manager_id is NULL
129     union all
130     select e.employee_id, e.first_name,e.last_name,e.manager_id, 1
131     from employees e
132     join employees eh on eh.manager_id =e.employee_id
133 )
134 select
135     employee_id,
136     concat(first_name, ' ', last_name) as employee_name,
137     level
138     from employee_hierarchy
139     order by level , employee_id
140
```

Result Grid | Filter Rows:  Export: Wrap Cell Content:

	employee_id	employee_name	level
▶	1	John Doe	0

## Activity:

```
1 •  create database activity;
2
3 •  use activity;
4
5      -- Sample Data Setup
6 •  CREATE TABLE employees (
7      employee_id INT PRIMARY KEY,
8      first_name VARCHAR(50),
9      last_name VARCHAR(50),
10     department VARCHAR(50),
11     salary DECIMAL(10, 2),
12     hire_date DATE
13 );
14
15 •  INSERT INTO employees VALUES
16     (1, 'John', 'Doe', 'IT', 75000, '2020-01-15'),
17     (2, 'Jane', 'Smith', 'HR', 65000, '2019-05-11'),
18     (3, 'Bob', 'Johnson', 'IT', 80000, '2018-03-23'),
19     (4, 'Alice', 'Williams', 'Finance', 72000, '2021-09-30'),
20     (5, 'Charlie', 'Brown', 'IT', 68000, '2022-02-14'),
21     (6, 'Eva', 'Davis', 'HR', 61000, '2020-11-18'),
22     (7, 'Frank', 'Miller', 'Finance', 79000, '2017-07-12'),
23     (8, 'Grace', 'Taylor', 'IT', 77000, '2019-04-22'),
24     (9, 'Henry', 'Anderson', 'Finance', 71000, '2021-01-05'),
25     (10, 'Ivy', 'Thomas', 'HR', 63000, '2022-06-30');
26
27 •  CREATE TABLE projects (
28     project_id INT PRIMARY KEY,
29     project_name VARCHAR(100),
```

```
30     start_date DATE,  
31     end_date DATE  
32 );  
33  
34 • INSERT INTO projects VALUES  
35     (1, 'Database Migration', '2023-01-01', '2023-06-30'),  
36     (2, 'New HR System', '2023-03-15', '2023-12-31'),  
37     (3, 'Financial Reporting Tool', '2023-02-01', '2023-11-30'),  
38     (4, 'IT Infrastructure Upgrade', '2023-05-01', '2024-04-30');  
39  
40 • CREATE TABLE employee_projects (  
41     employee_id INT,  
42     project_id INT,  
43     role VARCHAR(50),  
44     FOREIGN KEY (employee_id) REFERENCES employees(employee_id),  
45     FOREIGN KEY (project_id) REFERENCES projects(project_id)  
46 );  
47  
48 • INSERT INTO employee_projects VALUES  
49     (1, 1, 'Project Lead'),  
50     (2, 2, 'Project Manager'),  
51     (3, 1, 'Database Admin'),  
52     (4, 3, 'Financial Analyst'),  
53     (5, 4, 'Network Engineer'),  
54     (6, 2, 'HR Specialist'),  
55     (7, 3, 'Data Analyst'),  
56     (8, 4, 'Systems Architect'),  
57     (1, 4, 'Security Consultant'),  
58     (3, 4, 'Software Developer');
```

```

60      -- Questions
61
62      -- 1. Write a query to find the top 3 highest paid employees in each department.
63
64 •   SELECT e.department, e.first_name, e.last_name, e.salary
65     FROM employees e
66     WHERE (SELECT COUNT(DISTINCT salary)
67             FROM employees
68            WHERE department = e.department AND salary > e.salary) < 3
69     ORDER BY e.department, e.salary DESC;
70

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

	department	first_name	last_name	salary
▶	Finance	Frank	Miller	79000.00
	Finance	Alice	Williams	72000.00
	Finance	Henry	Anderson	71000.00
	HR	Jane	Smith	65000.00
	HR	Ivy	Thomas	63000.00
	HR	Eva	Davis	61000.00
	IT	Bob	Johnson	80000.00
	IT	Grace	Taylor	77000.00
	IT	John	Doe	75000.00

```

72      -- 2. Calculate the running total of salaries in each department, ordered by hire date.
73
74 •   SELECT e.department, e.first_name, e.last_name, e.salary,
75           SUM(e.salary) OVER (PARTITION BY e.department ORDER BY e.hire_date) AS running_total
76     FROM employees e
77     ORDER BY e.department, e.hire_date;
78
79

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

	department	first_name	last_name	salary	running_total
▶	Finance	Frank	Miller	79000.00	79000.00
	Finance	Henry	Anderson	71000.00	150000.00
	Finance	Alice	Williams	72000.00	222000.00
	HR	Jane	Smith	65000.00	65000.00
	HR	Eva	Davis	61000.00	126000.00
	HR	Ivy	Thomas	63000.00	189000.00
	IT	Bob	Johnson	80000.00	80000.00
	IT	Grace	Taylor	77000.00	157000.00
	IT	John	Doe	75000.00	232000.00
	IT	Charlie	Brown	68000.00	300000.00

```

-- 3. Find employees who are working on more than one project, along with the count of projects they're involved in.
81
82 • SELECT e.first_name, e.last_name, COUNT(ep.project_id) AS project_count
83   FROM employees e
84   JOIN employee_projects ep ON e.employee_id = ep.employee_id
85   GROUP BY e.employee_id
86   HAVING COUNT(ep.project_id) > 1;
87

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	project_count
John	Doe	2
Bob	Johnson	2

```
-- 4. Identify projects that have team members from all departments.
```

```

89
90 • SELECT p.project_name
91   FROM projects p
92   JOIN employee_projects ep ON p.project_id = ep.project_id
93   JOIN employees e ON ep.employee_id = e.employee_id
94   GROUP BY p.project_id
95   HAVING COUNT(DISTINCT e.department) = (SELECT COUNT(DISTINCT department) FROM employees);
96
97

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

project_name

```
-- 5. Calculate the average salary for each department, but only include employees hired in the last 3 years.
```

```

99
100 • SELECT department, AVG(salary) AS avg_salary
101   FROM employees
102   WHERE hire_date >= DATE_SUB(CURDATE(), INTERVAL 3 YEAR)
103   GROUP BY department;
104
105

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

department	avg_salary
IT	68000.000000
HR	63000.000000

```

106    -- 6. Create a pivot table showing the count of employees in each department, with columns for different salary ranges (e.g., <65000, 65000-75000, >75000)
107
108 •  SELECT department,
109      SUM(CASE WHEN salary < 65000 THEN 1 ELSE 0 END) AS "<65000",
110      SUM(CASE WHEN salary BETWEEN 65000 AND 75000 THEN 1 ELSE 0 END) AS "65000-75000",
111      SUM(CASE WHEN salary > 75000 THEN 1 ELSE 0 END) AS ">75000"
112  FROM employees
113  GROUP BY department;

```

Result Grid | Filter Rows: \_\_\_\_\_ | Export: \_\_\_\_\_ | Wrap Cell Content:

department	<65000	65000-75000	>75000
IT	0	2	2
HR	2	1	0
Finance	0	2	1

Result Grid Form Editor

```

115
116    -- 7. Find the employee(s) with the highest salary in their respective departments, who are also working on the longest-running project.
117
118 •  WITH longest_projects AS (
119     SELECT project_id, DATEDIFF(end_date, start_date) AS project_duration
120     FROM projects
121     ORDER BY project_duration DESC
122     LIMIT 1
123 )
124     SELECT e.first_name, e.last_name, e.salary, e.department
125     FROM employees e
126     JOIN employee_projects ep ON e.employee_id = ep.employee_id
127     JOIN longest_projects lp ON ep.project_id = lp.project_id
128     WHERE e.salary = (SELECT MAX(salary) FROM employees e2 WHERE e2.department = e.department);

```

Result Grid | Filter Rows: \_\_\_\_\_ | Export: \_\_\_\_\_ | Wrap Cell Content:

first_name	last_name	salary	department
Bob	Johnson	80000.00	IT

```

131    -- 8. Calculate the percentage of each department's salary compared to the total salary of the company.
132
133 •  SELECT department,
134     SUM(salary) AS department_total,
135     (SUM(salary) / (SELECT SUM(salary) FROM employees) * 100) AS percentage_of_total
136  FROM employees
137  GROUP BY department;

```

Result Grid | Filter Rows: \_\_\_\_\_ | Export: \_\_\_\_\_ | Wrap Cell Content:

department	department_total	percentage_of_total
IT	300000.00	42.194093
HR	189000.00	26.582278
Finance	222000.00	31.223629

```
-- 
141 -- 9. Identify employees who have a higher salary than their department's average, and show by what percentage their salary exceeds the average
142
143 • WITH department_avg AS (
144     SELECT department, AVG(salary) AS avg_salary
145     FROM employees
146     GROUP BY department
147 )
148     SELECT e.first_name, e.last_name, e.salary, da.avg_salary,
149         ((e.salary - da.avg_salary) / da.avg_salary * 100) AS percentage_above_avg
150     FROM employees e
151     JOIN department_avg da ON e.department = da.department
152     WHERE e.salary > da.avg_salary;
```

Result Grid | Filter Rows:  Export: Wrap Cell Content:

The screenshot shows a table with five columns: first\_name, last\_name, salary, avg\_salary, and percentage\_above\_avg. The data is as follows:

first_name	last_name	salary	avg_salary	percentage_above_avg
Jane	Smith	65000.00	63000.000000	3.1746031746
Bob	Johnson	80000.00	75000.000000	6.6666666667
Frank	Miller	79000.00	74000.000000	6.7567567568
Grace	Taylor	77000.00	75000.000000	2.6666666667

Result Grid | Form Editor | Field Types

```
-- 
155 -- 10. Create a query that shows a hierarchical view of employees and their projects, with multiple levels of projects if an employee is in more than one project
156
157 • SELECT e.first_name, e.last_name, ep.project_id, p.project_name, ep.role
158     FROM employees e
159     JOIN employee_projects ep ON e.employee_id = ep.employee_id
160     JOIN projects p ON ep.project_id = p.project_id
161     ORDER BY e.employee_id, ep.project_id;
```

Result Grid | Filter Rows:  Export: Wrap Cell Content:

The screenshot shows a table with five columns: first\_name, last\_name, project\_id, project\_name, and role. The data is as follows:

first_name	last_name	project_id	project_name	role
John	Doe	1	Database Migration	Project Lead
John	Doe	4	IT Infrastructure Upgrade	Security Consultant
Jane	Smith	2	New HR System	Project Manager
Bob	Johnson	1	Database Migration	Database Admin
Bob	Johnson	4	IT Infrastructure Upgrade	Software Developer
Alice	Williams	3	Financial Reporting Tool	Financial Analyst
Charlie	Brown	4	IT Infrastructure Upgrade	Network Engineer
Eva	Davis	2	New HR System	HR Specialist
Frank	Miller	3	Financial Reporting Tool	Data Analyst
Grace	Taylor	4	IT Infrastructure Upgrade	Systems Architect

Result Grid | Form Editor | Field Types

## Order of Execution of Queries:

1. From (including join)
2. Where
3. Group By
4. Having
5. Select
6. Order By
7. Top Limit

## Understanding and Interpreting EXPLAIN Output for Query Optimization:

### 1. id:

- Identifies each SELECT in the query
- A sequential number for simple queries
- Can be the same for JOINed tables or subqueries

### 2. select\_type:

- SIMPLE: Simple SELECT (no subqueries or UNIONs)
- PRIMARY: Outermost SELECT
- SUBQUERY: First SELECT in a subquery
- DERIVED: SELECT in a derived table (subquery in FROM clause)
- UNION: Second or later SELECT in a UNION
- UNION RESULT: Result of a UNION

### 3. table:

- The table this row refers to
- Can be a derived table name for subqueries

### 4. partitions:

- Partitions read, if the table is partitioned

### 5. type:

- Join type, crucial for optimization
- Common values:
  - system: Table has only one row
  - const: Matching one row, very fast
  - eq\_ref: One row read per combination of rows from previous tables
  - ref: All matching rows from an index read for each combination
  - range: Index used to retrieve rows within a range
  - index: Full index scan
  - ALL: Full table scan (slowest)

### 6. possible\_keys:

- 
- Indexes that could be used
  - NULL if no index could be used

7. key:

- The index actually chosen
- NULL if no index was used

8. key\_len:

- Length of the chosen key

9. ref:

- Columns or constants used with the key

10. rows:

- Estimated number of rows to examine

11. filtered:

- Estimated percentage of rows filtered by table condition

12. Extra:

- Additional information like "Using index", "Using temporary", "Using filesort"

Example:

Let's say we have this query:

```
sql
EXPLAIN SELECT e.first_name, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id
WHERE e.salary > 50000;
```

A possible EXPLAIN output might look like:

```

+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type  | possible_keys | key    | key_len | ref           | rows |
+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE     | e    | NULL    | ALL   | NULL        | NULL   | NULL    | NULL          |      |
1000 | 33.33 | Using where |
| 1 | SIMPLE     | d    | NULL    | eq_ref | PRIMARY    | PRIMARY | PRIMARY | 4            |
employees.department_id | 1 | 100.00 | NULL    |             |
+-----+-----+-----+-----+-----+-----+

```

Interpreting this:

1. It's a SIMPLE query (no subqueries or UNIONs).
2. It's scanning the entire employees table (type: ALL) which is inefficient.
3. It's using an index lookup on departments (type: eq\_ref).
4. It estimates examining 1000 rows from employees.
5. The WHERE condition is filtering about 33.33% of the rows.
6. It's using a WHERE clause on the employees table.

This output suggests that adding an index on the salary column in the employees table could potentially improve the query's performance.

Understanding EXPLAIN output helps in identifying performance bottlenecks and guides index creation and query rewriting for optimization.

## Detailed Breakdown of EXPLAIN Output Components for Query Optimization:

### 1. id:

- Purpose: Identifies each SELECT in the query.

- Example:

```
sql  
EXPLAIN  
SELECT * FROM employees  
UNION  
SELECT * FROM retired_employees;
```

Output:

id	select_type	table	...
1	PRIMARY	employees	...
2	UNION	retired_employees	...
3	UNION RESULT	<union1,2>	...

- Explanation: Here, 1 and 2 represent the two SELECT statements, while 3 is the result of the UNION.

### 2. select\_type:

- Purpose: Indicates the type of SELECT statement.

- Examples:

a) SIMPLE:

```
sql
```

```
EXPLAIN SELECT * FROM employees WHERE salary > 50000;
```

b) SUBQUERY:

sql

```
EXPLAIN SELECT * FROM employees WHERE department_id IN (SELECT id FROM departments WHERE location = 'New York');
```

c) DERIVED:

sql

```
EXPLAIN SELECT * FROM (SELECT id, name FROM employees WHERE salary > 50000) AS high_paid_employees;
```

3. table:

- Purpose: Shows which table the row refers to.

- Example:

sql

```
EXPLAIN SELECT e.name, d.name  
FROM employees e  
JOIN departments d ON e.department_id = d.id;
```

Output might show 'e' and 'd' in the table column.

4. partitions:

- Purpose: Indicates which partitions are being accessed.

- Example (assuming 'employees' is partitioned by year):

sql

```
EXPLAIN SELECT * FROM employees WHERE hire_date BETWEEN '2020-01-01' AND '2020-12-31';
```

Might show 'p2020' in the partitions column.

5. type:

- Purpose: Shows the join type, crucial for understanding query efficiency.

- Examples:

- a) const:

sql

```
EXPLAIN SELECT * FROM employees WHERE id = 1;
```

b) ref:

sql

```
EXPLAIN SELECT * FROM employees WHERE department_id = 5;
```

(Assuming department\_id is indexed but not unique)

c) range:

sql

```
EXPLAIN SELECT * FROM employees WHERE salary BETWEEN 50000 AND 60000;
```

(Assuming salary is indexed)

6. possible\_keys:

- Purpose: Lists indexes that could potentially be used.

- Example:

sql

```
EXPLAIN SELECT * FROM employees WHERE department_id = 5 AND salary > 50000;
```

Might show 'idx\_dept\_id, idx\_salary' if both columns are indexed.

7. key:

- Purpose: Shows the actual index used.

- Example: In the above query, it might show 'idx\_dept\_id' if MySQL decides this is more selective.

8. key\_len:

- Purpose: Indicates how much of the index is being used.

- Example: If department\_id is an INT (4 bytes), key\_len might show '4'.

9. ref:

- Purpose: Shows which columns or constants are compared to the index.

- Example:

sql

```
EXPLAIN SELECT * FROM employees e JOIN departments d ON e.department_id = d.id  
WHERE d.name = 'IT';
```

Might show 'const' for the departments table and 'd.id' for the employees table.

#### 10. rows:

- Purpose: Estimates the number of rows examined.
- Example: In a query on a large table, this might show a high number like 10000, indicating potential for optimization.

#### 11. filtered:

- Purpose: Estimates the percentage of rows filtered by table conditions.
- Example: In a query with WHERE salary > 50000, if 30% of employees meet this condition, it might show 30.00.

#### 12. Extra:

- Purpose: Provides additional information about how MySQL executes the query.

- Examples:

- a) "Using index" (good):

sql

```
EXPLAIN SELECT id, name FROM employees WHERE id > 1000;
```

- b) "Using temporary" (potential for optimization):

sql

```
EXPLAIN SELECT department_id, AVG(salary) FROM employees GROUP BY department_id ORDER BY AVG(salary);
```

- c) "Using filesort" (potential for optimization):

sql

```
EXPLAIN SELECT * FROM employees ORDER BY last_name;
```

(Assuming no index on last\_name)

Understanding these elements helps in query optimization. For instance, seeing 'ALL' in the type column or high numbers in the rows column often indicates areas for improvement, potentially by adding indexes or rewriting the query.

## Day 17 - DBMS Concepts

### 1. Primary Keys

A primary key is a column or set of columns in a table that uniquely identifies each row.

Key characteristics:

- Must be unique
- Cannot contain NULL values
- Should be immutable (not change over time)

Example:

Let's create a simple "Students" table:

```
sql
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DateOfBirth DATE
);
```

Here, StudentID is the primary key.

### 2. Foreign Keys

A foreign key is a column or set of columns in one table that refers to the primary key in another table. It establishes a link between two tables.

Key characteristics:

- Creates a relationship between tables
- Ensures referential integrity
- Can contain NULL values (unless specified otherwise)

Example:

Let's create a "Courses" table and an "Enrollments" table:

```
sql
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
```

```

CourseName VARCHAR(100)
);

CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

```

Here, StudentID and CourseID in the Enrollments table are foreign keys referencing the Students and Courses tables, respectively.

### 3. Normalization

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. There are several normal forms, but we'll focus on the first three, which are the most commonly used.

- First Normal Form (1NF):
  - Each column contains atomic (indivisible) values
  - No repeating groups

Example:

Consider this unnormalized table:

Students:

	StudentID   Name	Courses
1	John Doe	Math, Physics, Chemistry
2	Jane Smith	Biology, Chemistry

To bring it to 1NF:

```

sql
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100)
);

```

```
CREATE TABLE StudentCourses (
    StudentID INT,
    Course VARCHAR(50),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
);
```

- Second Normal Form (2NF):
  - Must be in 1NF
  - All non-key attributes are fully functionally dependent on the primary key

Example:

Consider this table that's in 1NF but not 2NF:

Orders:

OrderID | ProductID | ProductName | Quantity | CustomerID | CustomerName

To bring it to 2NF:

```
sql
CREATE TABLE Orders (
    OrderID INT,
    ProductID INT,
    Quantity INT,
    CustomerID INT,
    PRIMARY KEY (OrderID, ProductID)
);
```

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100)
);
```

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100)
);
```

- Third Normal Form (3NF):
  - Must be in 2NF
  - No transitive dependencies (non-key columns depend only on the primary key)

Example:

Consider this table that's in 2NF but not 3NF:

Employees:

EmployeeID | Name | DepartmentID | DepartmentName | ManagerID

To bring it to 3NF:

sql

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    DepartmentID INT,
    ManagerID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(100)
);
```

These normalized structures reduce data redundancy and improve data integrity. Each piece of information is stored in only one place, making updates and maintenance easier and reducing the risk of data inconsistencies.

Let's start with an unnormalized table for a small bookstore:

BookOrders:

OrderID | CustomerName | CustomerEmail | BookTitle | Author | Price | OrderDate | ShippingAddress

Step 1: First Normal Form (1NF)

To achieve 1NF, we need to ensure that:

1. Each column contains atomic values
2. There are no repeating groups

Our table is almost in 1NF, but let's assume that sometimes multiple books are ordered together. We'll split this into two tables:

```
sql
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerName VARCHAR(100),
    CustomerEmail VARCHAR(100),
    OrderDate DATE,
    ShippingAddress VARCHAR(200)
);

CREATE TABLE OrderDetails (
    OrderDetailID INT PRIMARY KEY,
    OrderID INT,
    BookTitle VARCHAR(200),
    Author VARCHAR(100),
    Price DECIMAL(10, 2),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);
```

Now we're in 1NF. Each column contains atomic values, and we've eliminated the repeating group (multiple books per order).

### Step 2: Second Normal Form (2NF)

To achieve 2NF:

1. The table must be in 1NF
2. All non-key attributes must be fully functionally dependent on the primary key

Our Orders table is already in 2NF because all attributes depend on the primary key OrderID.

However, in the OrderDetails table, BookTitle, Author, and Price don't depend on the full primary key (OrderDetailID), but only on part of it (BookTitle). Let's fix this:

```
sql
CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    BookTitle VARCHAR(200),
    Author VARCHAR(100),
    Price DECIMAL(10, 2)
);
```

```
CREATE TABLE OrderDetails (
    OrderDetailID INT PRIMARY KEY,
    OrderID INT,
    BookID INT,
    Quantity INT,
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID)
);
```

Now we're in 2NF. All non-key attributes in each table fully depend on their respective primary keys.

### Step 3: Third Normal Form (3NF)

To achieve 3NF:

1. The table must be in 2NF
2. There should be no transitive dependencies (non-key columns should depend only on the primary key)

In our current structure, we have a transitive dependency in the Orders table: CustomerEmail depends on CustomerName, which depends on OrderID. Let's resolve this:

```
sql
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100),
    CustomerEmail VARCHAR(100)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    ShippingAddress VARCHAR(200),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

-- Books and OrderDetails tables remain the same

Now we're in 3NF. All non-key attributes in each table depend only on the primary key, and we've eliminated transitive dependencies.

Final 3NF Structure:

```
sql
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100),
    CustomerEmail VARCHAR(100)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    ShippingAddress VARCHAR(200),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    BookTitle VARCHAR(200),
    Author VARCHAR(100),
    Price DECIMAL(10, 2)
);

CREATE TABLE OrderDetails (
    OrderDetailID INT PRIMARY KEY,
    OrderID INT,
    BookID INT,
    Quantity INT,
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID)
);
```

This structure in 3NF offers several benefits:

1. Reduced data redundancy: Customer and book information is stored only once.
2. Improved data integrity: Updating customer or book information only needs to happen in one place.
3. Easier data maintenance: Adding or modifying data is simpler and less error-prone.
4. Flexible querying: It's easier to query and analyze data across different aspects of the business.

Here's a summary of the changes made at each step:

- 
- 1NF: Split the single table into Orders and OrderDetails to handle multiple books per order.  
2NF: Extracted book information into a separate Books table to remove partial dependencies.  
3NF: Created a separate Customers table to eliminate transitive dependencies.

This example demonstrates how normalization progressively organizes data to reduce redundancy and improve data integrity. Each step builds on the previous one, resulting in a well-structured, efficient database design.

## ER Diagram

An Entity-Relationship Diagram (ERD) is a visual representation of the entities (objects or concepts) in a database and the relationships between them. It helps in designing and structuring a database by illustrating how data is interconnected.

Key Components of ER Diagrams:

1. Entities:
  - Represented by rectangles.
  - An entity can be a person, place, thing, or event.
  - Examples: **Customer**, **Order**, **Product**.
2. Attributes:
  - Represented by ovals connected to their respective entities.
  - Attributes provide additional information about the entity.
  - Examples: A **Customer** entity might have attributes like **CustomerID**, **Name**, and **Email**.
3. Relationships:
  - Represented by diamonds connecting entities.
  - Show how entities are related to each other.
  - Examples:
    - A **Customer** places an **Order** (One-to-Many relationship).
    - An **Order** contains one or more **Products** (Many-to-Many relationship).

Types of Relationships:

1. One-to-One (1:1):
  - An entity in A is associated with only one entity in B, and vice versa.
  - Example: Each **User** has one **Profile**.
2. One-to-Many (1):
  - An entity in A can be associated with multiple entities in B, but an entity in B is associated with only one entity in A.
  - Example: A **Customer** can place multiple **Orders**.

- 
3. Many-to-Many (M  
):
- Entities in A can be associated with multiple entities in B and vice versa.
  - Example: A **Student** can enroll in multiple **Courses**, and a **Course** can have multiple **Students**.

Example ERD:

For a simple eCommerce system, an ERD might include:

- Entities: **Customer**, **Order**, **Product**
- Relationships:
  - **Customer** places **Order** (1  
)
  - **Order** contains **Product** (M  
)

ERDs are useful for database design, ensuring that all necessary entities and relationships are considered before implementation.

## Specialization

Specialization is the process of defining sub-entities (subtypes) within a higher-level entity (supertype) based on distinct characteristics.

- Supertype: A generalized entity (e.g., **Vehicle**).
- Subtypes: Specific entities with unique attributes (e.g., **Car**, **Truck**, **Motorcycle**).
- ISA Relationship: Indicates that each subtype is a specific instance of the supertype (e.g., **Car ISA Vehicle**).
- Benefits:
  - Organizes data logically.
  - Reduces redundancy by inheriting common attributes.
  - Enhances data integrity by enforcing specific rules for subtypes.

Example:

- Supertype: **Employee**
  - Subtypes: **FullTimeEmployee**, **PartTimeEmployee**, **Contractor**.

## Aggregation

Aggregation is a modeling technique used to express a relationship between a whole and its parts. It allows for the representation of a relationship as a single abstract entity.

- **Whole-Part Relationship:** Represents a relationship where one entity (the whole) consists of multiple related entities (the parts).
- **Abstract Entity:** The aggregation forms a new entity that encapsulates the whole-part relationship, allowing it to be treated as a single entity in relationships with other entities.

**Benefits:**

- Simplifies complex relationships by treating a group of entities as a single unit.
- Helps in modeling scenarios where a relationship itself has attributes.

**Example:**

- **Entities:** **Project** (whole) and **Employee** (part).
- **Aggregation:** A **Project** consists of multiple **Employees**. The relationship can be represented as a single entity called **ProjectTeam** that encapsulates the relationship and its attributes (e.g., **startDate**, **endDate**).

## Generalization

Generalization is the opposite of specialization in database design and Entity-Relationship (ER) modeling. It involves creating a generalized supertype from multiple specific subtypes. This process focuses on identifying common attributes among several entities and abstracting them into a higher-level entity.

**Key Concepts:**

1. **Subtypes to Supertype:**
  - Generalization takes multiple specific entities (subtypes) that share common attributes and consolidates them into a single, more abstract entity (supertype).
  - Example: From **Car**, **Truck**, and **Motorcycle**, we can generalize to a single entity called **Vehicle**.
2. **Common Attributes:**
  - The supertype captures attributes that are common across the subtypes, reducing redundancy in the database schema.
  - Example: **Vehicle** may have attributes like **make**, **model**, and **year** that apply to all subtypes.
3. **IS-A Relationship:**
  - The relationship is often depicted as an "IS-A" relationship, where each subtype can be identified as a specific instance of the supertype.

- Example: **Car IS-A Vehicle.**
- 4. **Benefits:**
  - **Data Abstraction:** Simplifies the database design by abstracting common characteristics.
  - **Reduced Redundancy:** Common attributes are stored in one place (the supertype), minimizing duplication.
  - **Enhanced Maintenance:** Changes made to the supertype automatically propagate to all subtypes.

### **Example Scenario:**

Consider a **University Database**:

- **Subtypes:**
  - **UndergraduateStudent**
  - **GraduateStudent**
  - **Alumnus**
- **Generalized Supertype:**
  - **Student**
    - Attributes: **StudentID, Name, DateOfBirth.**

## Activity

```
1 •  create database activity;
2 •  use activity;
3
4  -- Create Tables
5 •  CREATE TABLE customers (
6      customer_id INT PRIMARY KEY AUTO_INCREMENT,
7      first_name VARCHAR(50),
8      last_name VARCHAR(50),
9      email VARCHAR(100) UNIQUE,
10     registration_date DATE
11 );
12
13 •  INSERT INTO customers (first_name, last_name, email, registration_date) VALUES
14     ('John', 'Doe', 'john.doe@example.com', '2023-01-15'),
15     ('Jane', 'Smith', 'jane.smith@example.com', '2023-02-10'),
16     ('Michael', 'Brown', 'michael.brown@example.com', '2023-03-22'),
17     ('Emily', 'Johnson', 'emily.johnson@example.com', '2023-05-05'),
18     ('David', 'Wilson', 'david.wilson@example.com', '2023-06-18');
19
20
21 •  CREATE TABLE products (
22     product_id INT PRIMARY KEY AUTO_INCREMENT,
23     product_name VARCHAR(100),
24     category VARCHAR(50),
25     price DECIMAL(10, 2),
26     stock_quantity INT
27 );
28
29 •  INSERT INTO products (product_name, category, price, stock_quantity) VALUES
```

```
30     ('Laptop', 'Electronics', 1200.00, 50),
31     ('Smartphone', 'Electronics', 800.00, 100),
32     ('Headphones', 'Accessories', 150.00, 200),
33     ('Desk Chair', 'Furniture', 250.00, 30),
34     ('Notebook', 'Stationery', 5.00, 500);
35
36
37 • Ⓜ CREATE TABLE orders (
38     order_id INT PRIMARY KEY AUTO_INCREMENT,
39     customer_id INT,
40     order_date DATETIME,
41     total_amount DECIMAL(10, 2),
42     FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
43 );
44
45 •   INSERT INTO orders (customer_id, order_date, total_amount) VALUES
46     (1, '2023-02-01 14:30:00', 1350.00),
47     (2, '2023-03-12 10:15:00', 805.00),
48     (3, '2023-04-01 16:45:00', 1250.00),
49     (1, '2023-05-15 09:10:00', 255.00),
50     (4, '2023-06-22 11:00:00', 1050.00);
51
52
53 • Ⓜ CREATE TABLE order_items (
54     order_item_id INT PRIMARY KEY AUTO_INCREMENT,
55     order_id INT,
56     product_id INT,
57     quantity INT,
58     price_per_unit DECIMAL(10, 2),
```

```

59     FOREIGN KEY (order_id) REFERENCES orders(order_id),
60     FOREIGN KEY (product_id) REFERENCES products(product_id)
61 );
62
63 • INSERT INTO order_items (order_id, product_id, quantity, price_per_unit) VALUES
64     (1, 1, 1, 1200.00), -- 1 Laptop
65     (1, 3, 1, 150.00), -- 1 Headphones
66     (2, 2, 1, 800.00), -- 1 Smartphone
67     (2, 5, 1, 5.00),   -- 1 Notebook
68     (3, 1, 1, 1200.00), -- 1 Laptop
69     (4, 3, 1, 150.00), -- 1 Headphones
70     (5, 2, 1, 800.00), -- 1 Smartphone
71     (5, 4, 1, 250.00); -- 1 Desk Chair
72
73
74 • CREATE TABLE product_reviews (
75     review_id INT PRIMARY KEY AUTO_INCREMENT,
76     product_id INT,
77     customer_id INT,
78     rating INT CHECK (rating BETWEEN 1 AND 5),
79     review_text TEXT,
80     review_date DATE,
81     FOREIGN KEY (product_id) REFERENCES products(product_id),
82     FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
83 );
84
85 • INSERT INTO product_reviews (product_id, customer_id, rating, review_text, review_date) VALUES
86     (1, 1, 5, 'Excellent laptop, highly recommended!', '2023-02-05'),
87     (3, 1, 4, 'Great sound quality, but a bit expensive.', '2023-02-07'),

```

```

88     (2, 2, 5, 'Fantastic smartphone with all the features I need.', '2023-03-15'),
89     (5, 2, 3, 'Just a regular notebook, nothing special.', '2023-03-18'),
90     (1, 3, 4, 'Good performance, but could be cheaper.', '2023-04-05');
91

```

```

92
93    -- Advanced SQL Practice Questions
94
95    -- Question 1: Find the top 3 customers who have spent the most money,
96    -- along with their total spend and the number of orders they've made.
97
98 •  SELECT c.customer_id, c.first_name, c.last_name,
99        SUM(o.total_amount) AS total_spent,
100       COUNT(o.order_id) AS num_orders
101   FROM customers c
102   JOIN orders o ON c.customer_id = o.customer_id
103   GROUP BY c.customer_id, c.first_name, c.last_name
104   ORDER BY total_spent DESC
105   LIMIT 3;
106

```

Result Grid					
	customer_id	first_name	last_name	total_spent	num_orders
▶	1	John	Doe	1605.00	2
	3	Michael	Brown	1250.00	1
	4	Emily	Johnson	1050.00	1

```

108    -- Question 2: Calculate the average rating for each product category,
109    -- but only include categories with at least 2 reviews.
110
111 •  SELECT p.category, AVG(r.rating) AS avg_rating
112   FROM products p
113   JOIN product_reviews r ON p.product_id = r.product_id
114   GROUP BY p.category
115   HAVING COUNT(r.review_id) >= 2;
116

```

Result Grid		
	category	avg_rating
▶	Electronics	4.6667

```
...
118      -- Question 3: Find products that have never been ordered.
119
120 •  SELECT p.product_id, p.product_name
121   FROM products p
122   LEFT JOIN order_items oi ON p.product_id = oi.product_id
123 WHERE oi.order_item_id IS NULL;
124
125
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
product_id	product_name			

```

125
126    -- Question 4: For each customer, find the time difference between their
127    -- registration date and their first order date.
128
129 •  SELECT c.customer_id, c.first_name, c.last_name,
130        DATEDIFF(MIN(o.order_date), c.registration_date) AS days_to_first_order
131    FROM customers c
132    JOIN orders o ON c.customer_id = o.customer_id
133    GROUP BY c.customer_id, c.first_name, c.last_name;
134

```

	customer_id	first_name	last_name	days_to_first_order
▶	1	John	Doe	17
	2	Jane	Smith	30
	3	Michael	Brown	10
	4	Emily	Johnson	48

```

135
136    -- Question 5: Create a report showing the total revenue for each month,
137    -- along with a running total of revenue throughout the year.
138
139 •  SELECT DATE_FORMAT(o.order_date, '%Y-%m') AS order_month,
140        SUM(o.total_amount) AS monthly_revenue,
141        SUM(SUM(o.total_amount)) OVER (ORDER BY DATE_FORMAT(o.order_date, '%Y-%m')) AS running_total
142    FROM orders o
143    GROUP BY order_month;
144
145

```

	order_month	monthly_revenue	running_total
▶	2023-02	1350.00	1350.00
	2023-03	805.00	2155.00
	2023-04	1250.00	3405.00
	2023-05	255.00	3660.00
	2023-06	1050.00	4710.00

```
145
146 -- Question 6: Identify customers who have made a purchase but have never left a product review.
147
148 • SELECT c.customer_id, c.first_name, c.last_name
149   FROM customers c
150   JOIN orders o ON c.customer_id = o.customer_id
151   LEFT JOIN product_reviews r ON c.customer_id = r.customer_id
152   WHERE r.review_id IS NULL
153   GROUP BY c.customer_id, c.first_name, c.last_name;
```

Result Grid			
	customer_id	first_name	last_name
▶	4	Emily	Johnson

```
155
156 -- Question 7: Find the product that has been ordered the most times (by quantity).
157
158 • SELECT p.product_id, p.product_name, SUM(oi.quantity) AS total_quantity_ordered
159   FROM products p
160   JOIN order_items oi ON p.product_id = oi.product_id
161   GROUP BY p.product_id, p.product_name
162   ORDER BY total_quantity_ordered DESC
163   LIMIT 1;
```

Result Grid			
	product_id	product_name	total_quantity_ordered
▶	1	Laptop	2

```

165
166    -- Question 8: Calculate the percentage of total revenue that each product category contributes.
167
168 •   SELECT p.category,
169        SUM(oi.quantity * oi.price_per_unit) AS category_revenue,
170        (SUM(oi.quantity * oi.price_per_unit) / (SELECT SUM(quantity * price_per_unit) FROM order_items) * 100) AS percentage_revenue
171    FROM products p
172    JOIN order_items oi ON p.product_id = oi.product_id
173    GROUP BY p.category;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	category	category_revenue	percentage_revenue
▶	Electronics	4000.00	87.815587
	Accessories	300.00	6.586169
	Furniture	250.00	5.488474
	Stationery	5.00	0.109769

```

176    -- Question 9: For each customer, find their most frequently purchased product category.
177

```

```

178 •   WITH CategoryPurchases AS (
179     SELECT c.customer_id, p.category, SUM(oi.quantity) AS total_quantity
180     FROM customers c
181     JOIN orders o ON c.customer_id = o.customer_id
182     JOIN order_items oi ON o.order_id = oi.order_id
183     JOIN products p ON oi.product_id = p.product_id
184     GROUP BY c.customer_id, p.category
185   )
186     SELECT customer_id, category
187   FROM (
188     SELECT customer_id, category,
189           ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY total_quantity DESC) AS row_num
190     FROM CategoryPurchases
191   ) AS ranked_categories
192   WHERE row_num = 1;
193

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	customer_id	category
▶	1	Accessories
	2	Electronics
	3	Electronics
	4	Electronics

```

194
195 -- Question 10: Create a query to show the distribution of ratings (count of 1-star, 2-star, etc.) for each product.
196
197 • SELECT p.product_name, r.rating, COUNT(r.review_id) AS rating_count
198   FROM products p
199   JOIN product_reviews r ON p.product_id = r.product_id
200   GROUP BY p.product_name, r.rating
201   ORDER BY p.product_name, r.rating;

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

product_name	rating	rating_count
Headphones	4	1
Laptop	4	1
Laptop	5	1
Notebook	3	1
Smartphone	5	1

**Creating an ER (Entity-Relationship) diagram, normalization, and generalization for above database schema and data:**

## 1. ER Diagram:

The ER diagram helps represent the relationships between entities in your database. Based on the schema, the key entities and relationships are:

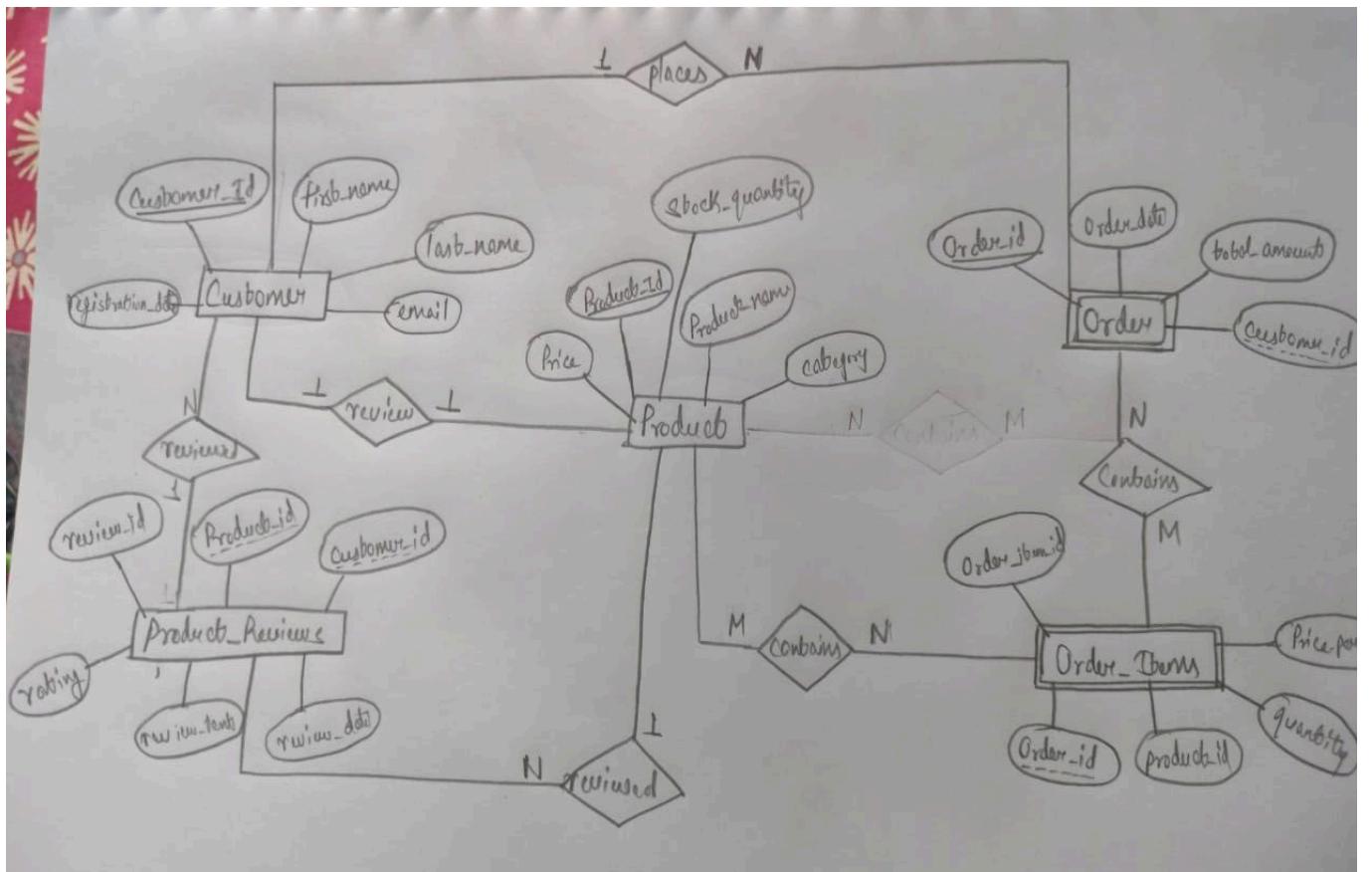
Entities:

- Customers:
  - Attributes: `customer_id`, `first_name`, `last_name`, `email`, `registration_date`
- Products:
  - Attributes: `product_id`, `product_name`, `category`, `price`, `stock_quantity`
- Orders:
  - Attributes: `order_id`, `customer_id` (foreign key), `order_date`, `total_amount`
- Order\_Items:
  - Attributes: `order_item_id`, `order_id` (foreign key), `product_id` (foreign key), `quantity`, `price_per_unit`
- Product\_Reviews:
  - Attributes: `review_id`, `product_id` (foreign key), `customer_id` (foreign key), `rating`, `review_text`, `review_date`

Relationships:

- Customers place Orders:

- One customer can place many orders (1 relationship).
- Orders contain Products (through Order\_Items):
  - Each order can have multiple products, and each product can appear in multiple orders (M relationship). The **Order\_Items** table represents this relationship.
- Customers review Products:
  - A customer can write many reviews for different products, but typically only one review per product (1 relationship).
- Products are reviewed by Customers:
  - A product can have multiple reviews from different customers (1 relationship).



---

## 2. Normalization:

Normalization is the process of organizing the database to reduce redundancy and improve data integrity. This schema appears to already be in **3rd Normal Form (3NF)**, which includes:

### 1NF: (Eliminates repeating groups)

- Each table has atomic columns, meaning no columns contain multiple values (such as lists or arrays).
- All entries in columns are of the same type.

### 2NF: (Eliminates partial dependencies)

- All non-primary key attributes are fully dependent on the primary key in each table.
  - In `orders`, all columns (`customer_id`, `order_date`, `total_amount`) are dependent on `order_id`.
  - In `order_items`, the `quantity` and `price_per_unit` are dependent on both `order_id` and `product_id`.

### 3NF: (Eliminates transitive dependencies)

- No transitive dependencies exist between non-primary key attributes.
  - For instance, in the `customers` table, `email` and `registration_date` depend only on `customer_id`.

## 3. Generalization:

Generalization in databases is a way of simplifying relationships between entities by identifying common features among different entities and grouping them into higher-level entities.

In this case, there is no immediate need for generalization because the entities represent specific real-world objects (customers, orders, products) with clear distinctions.

## Stock Price Analysis Using Window Functions in SQL [Lag( ) and Lead( )]:

```
205
206      -- Create a sample table for stock prices
207 • CREATE TABLE stock_prices (
208     date DATE,
209     stock_symbol VARCHAR(10),
210     closing_price DECIMAL(10, 2)
211 );
212
213
214      -- Insert sample data
215 • INSERT INTO stock_prices (date, stock_symbol, closing_price) VALUES
216     ('2023-01-01', 'AAPL', 150.00),
217     ('2023-01-02', 'AAPL', 152.50),
218     ('2023-01-03', 'AAPL', 151.75),
219     ('2023-01-04', 'AAPL', 153.00),
220     ('2023-01-05', 'AAPL', 155.25),
221     ('2023-01-01', 'GOOGL', 2800.00),
222     ('2023-01-02', 'GOOGL', 2825.00),
223     ('2023-01-03', 'GOOGL', 2815.50),
224     ('2023-01-04', 'GOOGL', 2830.00),
225     ('2023-01-05', 'GOOGL', 2850.75);
---
```

```
228 • select date,  
229     stock_symbol,  
230     closing_price,  
231     lead(closing_price,2) over (partition by stock_symbol order by date) as next_day_price  
232     from stock_prices  
233     order by stock_symbol, date;  
234  
235
```

Result Grid				
	date	stock_symbol	closing_price	next_day_price
▶	2023-01-01	AAPL	150.00	151.75
	2023-01-02	AAPL	152.50	153.00
	2023-01-03	AAPL	151.75	155.25
	2023-01-04	AAPL	153.00	NULL
	2023-01-05	AAPL	155.25	NULL
	2023-01-01	GOOGL	2800.00	2815.50
	2023-01-02	GOOGL	2825.00	2830.00
	2023-01-03	GOOGL	2815.50	2850.75
	2023-01-04	GOOGL	2830.00	NULL
	2023-01-05	GOOGL	2850.75	NULL

```
237 • select date,  
238     stock_symbol,  
239     lag(closing_price,1) over (partition by stock_symbol order by date) as previous_day_price  
240     ,closing_price  
241     from stock_prices  
242     order by stock_symbol, date;  
243
```

Result Grid				
	date	stock_symbol	previous_day_price	closing_price
▶	2023-01-01	AAPL	NULL	150.00
	2023-01-02	AAPL	150.00	152.50
	2023-01-03	AAPL	152.50	151.75
	2023-01-04	AAPL	151.75	153.00
	2023-01-05	AAPL	153.00	155.25
	2023-01-01	GOOGL	NULL	2800.00
	2023-01-02	GOOGL	2800.00	2825.00
	2023-01-03	GOOGL	2825.00	2815.50
	2023-01-04	GOOGL	2815.50	2830.00
	2023-01-05	GOOGL	2830.00	2850.75

```
246 • select date,  
247   stock_symbol,  
248   closing_price,  
249   lag(closing_price,1) over (partition by stock_symbol order by date) as previous_day_price  
250 ,closing_price-LAG(closing_price,1) over (partition by stock_symbol order by date) as price_change  
251 from stock_prices  
252 order by stock_symbol, date;  
253
```

Result Grid					
	date	stock_symbol	closing_price	previous_day_price	price_change
▶	2023-01-01	AAPL	150.00	NULL	NULL
	2023-01-02	AAPL	152.50	150.00	2.50
	2023-01-03	AAPL	151.75	152.50	-0.75
	2023-01-04	AAPL	153.00	151.75	1.25
	2023-01-05	AAPL	155.25	153.00	2.25
	2023-01-01	GOOGL	2800.00	NULL	NULL
	2023-01-02	GOOGL	2825.00	2800.00	25.00
	2023-01-03	GOOGL	2815.50	2825.00	-9.50
	2023-01-04	GOOGL	2830.00	2815.50	14.50
	2023-01-05	GOOGL	2850.75	2830.00	20.75

# Day 18 - Database Concepts

## MySQL Execution Engine

The MySQL execution engine is responsible for executing SQL queries and returning results to the client. It works as part of MySQL's server architecture, and its main components are:

1. Parser: Converts the SQL query into a parse tree.
2. Optimizer: Determines the most efficient way to execute the query by considering factors like indexes, join methods, and query rewriting.
3. Executor: Executes the optimized query plan, accessing tables and indexes to retrieve or modify data as specified by the query.
4. Storage Engine: Handles the interaction with physical data storage, using engines like InnoDB, MyISAM, etc., which manage table formats and file I/O.

### A. Parser

The Parser in MySQL is a component responsible for interpreting and validating SQL queries. It converts a raw SQL query into a structure that the database can process, typically known as a parse tree or syntax tree.

Key Functions of the Parser:

1. Syntax Analysis:
  - The parser first checks if the SQL query is syntactically correct according to the SQL grammar. It ensures the query structure follows the correct format, like `SELECT ... FROM ... WHERE ...`.
  - If there are syntax errors, the parser throws an error before moving to any further processing.
2. Tokenization:
  - The parser breaks the query into tokens (small, meaningful units like keywords, table names, operators, etc.). For example, in `SELECT name FROM users`, the tokens would be `SELECT`, `name`, `FROM`, and `users`.
3. Parse Tree Generation:
  - After tokenizing, the parser generates a parse tree, which is a tree-like representation of the query structure. Each node in the tree represents a different component of the SQL statement (e.g., SELECT, WHERE, JOIN).
  - This tree helps the next stages (like optimization) understand the logical flow of the query.
4. Semantic Checking:

- Beyond checking syntax, the parser may also perform semantic checks. This ensures that the query makes sense logically, like verifying that the tables and columns mentioned exist, or that data types are compatible.
5. Pass to the Optimizer:
- Once the parsing is complete and the query is valid, the parse tree is passed to the optimizer for further processing and execution planning.

## B. Optimizer

The Optimizer in MySQL is the component responsible for determining the most efficient way to execute a given SQL query. After the parser generates the parse tree from the SQL query, the optimizer analyzes it and decides on the best execution strategy.

Key Functions of the Optimizer:

1. Execution Plan Generation:
  - The optimizer creates multiple possible execution plans for a query, which are different ways the query could be executed. Each plan involves different methods of accessing and joining tables, using indexes, and applying filters.
2. Cost-Based Optimization:
  - MySQL uses a cost-based optimizer, meaning it evaluates the "cost" of each execution plan in terms of CPU, memory, and I/O usage. It then selects the plan with the lowest cost, which is expected to be the most efficient.
  - The cost estimation is based on several factors like:
    - Table size.
    - Number of rows to scan.
    - Whether indexes can be used.
    - How much data needs to be read from disk.
3. Index Selection:
  - The optimizer decides whether to use an index for faster data retrieval. It evaluates which available indexes (if any) will speed up data lookups and joins.
  - It also decides when to perform full table scans if that is more efficient for small datasets or unindexed queries.
4. Join Optimization:
  - For queries that involve multiple tables, the optimizer selects the join order and the join type (e.g., nested loops, merge joins, or hash joins). This decision minimizes the time needed to match rows between tables.

- It considers factors like which table is smaller, which has an index, and how many rows will be joined.
5. Query Rewriting:
- The optimizer can rewrite certain parts of the query to make it more efficient. For example, it may transform subqueries into joins or push down conditions (filters) to lower levels in the execution plan, reducing the amount of data processed early on.
6. Subquery Optimization:
- Subqueries, especially correlated subqueries, can be inefficient. The optimizer may decide to convert subqueries into JOIN operations or other more efficient structures to improve performance.
7. Caching & Reuse:
- The optimizer may also use query caching if similar queries have been executed recently, allowing it to reuse the execution plan.

## C. Query Hinting

Query hinting in MySQL allows developers to influence the decisions of the query optimizer by explicitly specifying hints in SQL queries. These hints guide the optimizer to use a particular strategy (like choosing specific indexes, join orders, or access methods) that the developer believes will result in better query performance.

## D. Storage Engine

A Storage Engine in MySQL is the component responsible for managing how data is stored, retrieved, and updated in the database. MySQL supports multiple storage engines, and each one handles data differently, offering varying levels of performance, reliability, and feature sets (e.g., support for transactions, foreign keys, or different types of indexing).

### Key Storage Engines in MySQL:

1. InnoDB (Default Engine):
  - Transactions: Supports ACID-compliant transactions, meaning it guarantees atomicity, consistency, isolation, and durability.
  - Row-level Locking: Provides row-level locking for better performance in high-concurrency environments.
  - Foreign Keys: Supports foreign key constraints for referential integrity.

- Crash Recovery: Has built-in crash recovery features using the transaction log, making it highly reliable.
  - Indexes: Uses a clustered index for primary keys, which stores data physically according to the key.
2. Use Case: Best suited for transactional applications that require high data integrity and consistency, such as banking systems or e-commerce platforms.
  3. MyISAM:
    - Non-Transactional: Does not support transactions, so it's faster for read-heavy operations but lacks ACID properties.
    - Table-level Locking: Uses table-level locking, which can cause bottlenecks in environments with many concurrent writes.
    - Smaller Footprint: Requires less disk space than InnoDB, as it doesn't store as much metadata.
    - Full-text Indexing: Offers built-in full-text search support, which can be useful for searching text-heavy content.
    - No Foreign Keys: Lacks support for foreign key constraints.

### **Query Execution Methods:**

In MySQL, query execution methods refer to the various strategies used by the database engine to retrieve, modify, or manage data efficiently. After parsing and optimizing a query, the MySQL executor uses different methods to actually carry out the query. These methods vary based on the type of query and the available resources, such as indexes, joins, and access patterns.

Here are some common query execution methods:

#### **1. Full Table Scan**

- Description: MySQL reads all rows in the table to find the relevant ones that match the query's conditions. It scans the entire table even if only a few rows match.

#### **2. Index Scan**

- Description: Instead of reading the entire table, MySQL reads data from an index to satisfy the query conditions. This reduces the number of rows that need to be scanned.

#### **3. Index Seek (Direct Lookup)**

- Description: MySQL performs a direct lookup in an index for specific values. It jumps directly to the index entry that satisfies the query.

---

#### 4. Range Scan

- Description: MySQL uses an index to retrieve rows within a specified range. It scans the portion of the index that falls within the given range.

#### 5. Index-Only Scan (Covering Index)

- Description: MySQL retrieves all the data required for the query directly from the index, without having to access the actual table.

#### 6. Join Methods

- Nested Loop Join:
  - Description: For each row in the first (outer) table, MySQL checks matching rows in the second (inner) table.

### **Query Cache:**

Query Cache in MySQL is a feature that stores the result of a SELECT query and reuses that result when an identical query is executed later. This can significantly improve performance for repeated queries by avoiding the need to execute the same query multiple times. Instead, MySQL retrieves the result directly from the cache, reducing query execution time.

## Database Performance Testing with Large Data Insertion, Indexing, and Joins:

```
1   create database test_db_poc;
2
3 •  use test_db_poc;
4 •  create table if not exists large_table(
5     id int auto_increment primary key,
6     name varchar(50),
7     value INT
8 );
9
10  DELIMITER //
11 •  CREATE procedure INSERT_MILLION_RECORDS()
12  BEGIN
13      declare i int default 0;
14      while i <100000 do
15          insert into large_table(name,value) values(concat('Name',i),floor(1 + RAND()*1000000));
16          set i=i+1;
17      end while;
18  END //
19
20  DELIMITER ;
-- 
22 •  SHOW procedure status WHERE DB ='test_db_poc';
23 |
```

Result Grid									
Db	Name	Type	Definer	Modified	Created	Security_type	Comment	character_set_client	collation_connection
test_db_poc	INSERT_MILLION_RECORDS	PROCEDURE	root@localhost	2024-10-10 10:04:20	2024-10-10 10:04:20	DEFINER		utf8mb4	utf8mb4_0900_ai_ci

```
24 •  select count(*) from large_table;
```

```
25
```

Result Grid	
Filter Rows:	
	count(*)
▶	38362

```
29 •  select sql_no_cache sum(value) from large_table ;
```

```
30 •  select sum(value) from large_table;
```

Result Grid	
Filter Rows:	
	sum(value)
▶	25252629957

```
30 •   select sum(value) from large_table; |
```

```
31
```

Result Grid |



Filter Rows:

Export:



Wrap C

sum(value)
▶ 26738063054

```
32 •   create index idx_value on large_table(value);
```

```
33 •   alter table large_table drop index idx_value;
```

```
34 •   select * from large_table;
```

```
35
```

Result Grid |



Filter Rows:

Edit:



Export/Im

id	name	value
▶ 1	Name0	266249
2	Name1	135184
3	Name2	877173
4	Name3	980313
5	Name4	270045
6	Name5	409284
7	Name6	226297

large\_table 6 ×

```
36 •   select a.value
```

```
37     from large_table a
```

```
38     cross join large_table b
```

```
39     on a.value=b.value*100;
```

Result Grid |



Filter Rows:

Exp

value
▶ 443200
578700
507100
910100
499500
427500
310000

Result 7 ×

---

## **Detailed Flow of Query Execution from Application to SSD Disk and Data Retrieval:**

Query: SELECT name FROM students WHERE id = 1008;

### 1. Database Layer:

- Determines student 1008 is in Page 1
- Page 1 = offset 8192, length 8192 bytes in students table file

### 2. File System Layer:

- Translates to reading two 4KB blocks:  
Block 2 (offset 8192-12287)  
Block 3 (offset 12288-16383)

### 3. LBA Translation:

- Assuming students table starts at LBA 1000000:  
Block 2 → LBA 1000002  
Block 3 → LBA 1000003

### 4. SSD Controller:

- Receives command: Read LBAs 1000002 and 1000003
- Maps to SSD pages:  
LBA 1000002 → SSD Page 500000, offset 0  
LBA 1000003 → SSD Page 500000, offset 4096

### 5. Physical Storage:

- Reads entire SSD Page 500000 (16KB)

### 6. Data Return Path:

- SSD Controller extracts relevant 8KB from Page 500000
- File System reassembles into original 8KB request
- Database receives 8KB page, finds student 1008, returns name



---

## **Let's use a large library as our analogy:**

Database (Library Catalog):

Imagine a library catalog system. It doesn't store the actual books; instead, it stores information about where to find each book.

Example:

Book Title: "Database Systems"

Location: "Floor 3, Shelf 24, Position 15"

This is similar to how a database stores offsets instead of actual data. The offset is like saying "Floor 3, Shelf 24, Position 15" instead of storing the entire book in the catalog.

File System (Library Floors and Shelves):

The file system is like the physical organization of the library. It doesn't store the books either, but it manages how the shelves are arranged and numbered.

Example:

Floor 3 has shelves numbered 1-50

Each shelf can hold 100 books

LBA (Logical Book Address):

Now, imagine the library uses a unique numbering system for all books, regardless of their physical location. This is like an LBA.

Example:

"Database Systems" might be assigned LBA 10045

Physical Storage (Actual Book Location):

This is where the book actually sits on the shelf.

Now, let's see how this works in practice:

You ask the librarian (database query) for "Database Systems".

The librarian checks the catalog (database) and finds: "Floor 3, Shelf 24, Position 15".

This translates to a file offset in the computer world.

The librarian then uses a master list (file system) that says: "Floor 3, Shelf 24 = LBA range 10000-10099".

The LBA for this book is calculated as 10000 (start of shelf) + 15 (position) = 10015.

A robot (SSD controller) that only understands LBAs is sent to fetch book 10015. The robot has its own map that translates LBA 10015 to an exact physical location. The book is retrieved and brought back to you.

Why is this system used?

Flexibility: The library can reorganize its shelves without changing the catalog.

Efficiency: It's faster to look up a simple number (LBA) than a complex address.

Abstraction: The librarian doesn't need to know how the robot finds the books.

In computers:

Database offset: Like the catalog entry, it tells where in a file to find data.

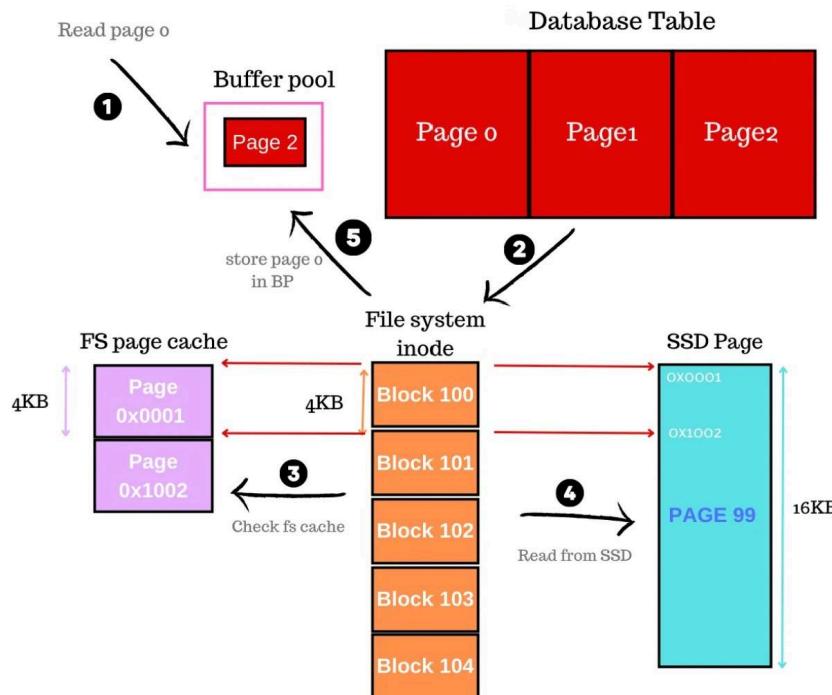
File system: Manages how files are organized on the storage device.

LBA: A simple numbering system for all data blocks on a storage device.

Physical storage: The actual location on the SSD or hard drive.

This system allows each layer (database, file system, storage device) to work independently while still communicating effectively.

## Following a database read to the metal



## Customer and Order Data Management: Table Creation, Data Population, and Query Optimization:

```
1 •  create database customer;
2 •  use customer;
3
4 •  create table customers(
5     customer_id int primary key,
6     name varchar(100),
7     email varchar(100),
8     registration date);
9
10 • select * from customers;
11
12 • create index idx_customer_reg_Date on customers(registration);
13 • update customers set registration='2024-0-20';
14
15 • set session join_buffer_size=4194304;
16 • explain select o.orderId,o.orderDate,o.TotalAmount,c.Name
   from orders o
   join customers c  use index(idx_customer_reg_Date) on o.customer_id =c.customer_id
   where c.registration>= DATE_SUB(CURDATE(),INTERVAL 30 day);
17
18
19
20
21
22
23 -- Populate Customers table
24 • INSERT INTO Customers (Customer_ID, Name, Email, registration)
25 VALUES
26 (1, 'John Doe', 'john.doe@email.com', '2024-09-15'),
27 (2, 'Jane Smith', 'jane.smith@email.com', '2024-09-20'),
28 (3, 'Bob Johnson', 'bob.johnson@email.com', '2024-09-25'),
29 (4, 'Alice Brown', 'alice.brown@email.com', '2024-09-30'),
30 (5, 'Charlie Davis', 'charlie.davis@email.com', '2024-10-01'),
31 (6, 'Eva Wilson', 'eva.wilson@email.com', '2024-10-02'),
32 (7, 'Frank Miller', 'frank.miller@email.com', '2024-10-03'),
33 (8, 'Grace Lee', 'grace.lee@email.com', '2024-10-04'),
34 (9, 'Henry Taylor', 'henry.taylor@email.com', '2024-10-05'),
35 (10, 'Ivy Clark', 'ivy.clark@email.com', '2024-10-06');
36
37 -- Insert 90 more customers with registration dates spread over the last 60 days
38 • INSERT INTO Customers (Customer_ID, Name, Email, registration)
39 SELECT
40     10 + num,
41     CONCAT('Customer', num),
42     CONCAT('customer', num, '@email.com'),
43     DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND() * 60) DAY)
44 FROM (
45     SELECT a.N + b.N * 10 + 1 as num
46     FROM
47         (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION
48         (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8) b
49     ORDER BY num
50     LIMIT 90
51 ) numbers;
52 •  create table orders(orderid int primary key,
53     customer_id int,
54     orderdate date,
55     totalamount decimal(10,2),
56     foreign key(customer_id) references customers(customer_id));
57
58
```

```

--  

59  -- Populate Orders table  

60 • INSERT INTO Orders (OrderID, customer_id, OrderDate, TotalAmount)  

61  VALUES  

62  (1, 1, '2024-09-16', 150.00),  

63  (2, 2, '2024-09-21', 200.50),  

64  (3, 3, '2024-09-26', 75.25),  

65  (4, 4, '2024-10-01', 300.75),  

66  (5, 5, '2024-10-02', 50.00),  

67  (6, 6, '2024-10-03', 125.50),  

68  (7, 7, '2024-10-04', 80.00),  

69  (8, 8, '2024-10-05', 220.25),  

70  (9, 9, '2024-10-06', 175.00),  

71  (10, 10, '2024-10-07', 90.50);  

72  

73  -- Insert 190 more orders with random customers and dates  

74 • INSERT INTO Orders (OrderID, Customer_ID, OrderDate, TotalAmount)  

75  SELECT  

76    10 + num,  

77    1 + FLOOR(RAND() * 100),  

78    DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND() * 30) DAY),  

79    ROUND(50 + RAND() * 450, 2)  

80  FROM ( SELECT a.N + b.N * 10 + c.N * 100 + 1 AS num  

81    FROM  

82      (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION  

83      (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION  

84      (SELECT 0 AS N UNION SELECT 1) c  

85    ORDER BY num  

86    LIMIT 190  

87  ) numbers;  


```

Result Grid | Filter Rows:  Export: Wrap Cell Content:

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	c	NULL	ALL	idx_customer_reg_Date	NULL	NULL	NULL	100	51.00	Using where
	1	SIMPLE	o	NULL	ref	customer_id	customer_id	5	customer.c.customer_id	1	100.00	NULL

# Day 19 - Relational Database Design Concepts

## Database Design Principles: From Normalization to Data Warehousing:

### 1. Database Design Modeling

Database design modeling involves structuring and defining the schema of the database, which includes defining tables, columns, data types, relationships, and constraints. Two common approaches are:

- Relational Data Modeling: Used for transactional systems where data is stored in tables (relations). Key principles include normalization and ensuring consistency through constraints.
- Dimensional Data Modeling: Typically used for analytical or reporting purposes, where data is structured in a way that facilitates fast querying and reporting (common in data warehousing).

### 2. Normalization

Normalization is a method for organizing data in a database to reduce redundancy and improve data integrity. The main focus of normalization is ensuring that each piece of data is stored once. Key forms of normalization include:

- 1NF (First Normal Form): Ensure that the values in each column of a table are atomic (indivisible).
- 2NF (Second Normal Form): Every non-primary key attribute must depend fully on the primary key, not a subset of it.
- 3NF (Third Normal Form): Eliminate transitive dependencies, where non-key attributes are dependent on other non-key attributes.
- Boyce-Codd Normal Form (BCNF): A stronger version of 3NF, dealing with certain edge cases where 3NF may not be sufficient.

Benefits of Normalization:

- Eliminates redundant data.
- Ensures data consistency and integrity.
- Simplifies maintenance and updates.

Trade-offs:

- Increased complexity for queries due to multiple table joins.

- May not perform well for analytical queries (hence the need for denormalization in certain scenarios).

### 3. Dimensional Modeling

Dimensional modeling is a design methodology used primarily for data warehousing and business intelligence applications, where the goal is to optimize for fast query performance, especially for reporting and analytics. The two main components of dimensional modeling are:

- Fact Tables: Store measurable, quantitative data (e.g., sales, revenue). Each row represents a specific event or transaction.
- Dimension Tables: Store descriptive, categorical information about the data in fact tables (e.g., time, product, customer).

Star Schema:

- A common design in dimensional modeling where a central fact table is surrounded by dimension tables, forming a star-like structure.

Snowflake Schema:

- A variation of the star schema where dimension tables are further normalized, breaking them into multiple related tables, resulting in a "snowflake" shape.

### 4. Data Warehouse

A data warehouse is a large, centralized repository that aggregates data from multiple sources, optimized for querying and reporting rather than transactional processing. It stores historical data and supports complex queries for analytical purposes.

Key characteristics of a data warehouse include:

- Subject-Oriented: Data is organized around specific business domains (e.g., sales, finance).
- Integrated: Data is consolidated from various sources into a cohesive format.
- Time-Variant: Data is stored with historical context, allowing for trend analysis.
- Non-Volatile: Once data is entered, it is not typically updated or deleted.

Data warehouses often use dimensional modeling to make querying data faster and easier for analysts, focusing on OLAP (Online Analytical Processing) rather than OLTP (Online Transaction Processing).

## **Quantitative Data in Centralized Repositories:**

When working with a centralized data repository (such as a data warehouse), **quantitative data** is key for generating business intelligence reports, financial analysis, and other critical metrics. The repository typically contains:

- **Transaction Data:** Data like sales, revenue, and costs, all of which are numerical and subject to aggregation and analysis.
- **Fact Tables:** Storing quantitative data such as sales amounts or product counts, which can be summarized and analyzed across different dimensions (e.g., time, geography, product).

## **Approaches for organizing data in data warehousing:**

### **Star Schema**

Overview:

- A star schema consists of a central fact table connected to multiple dimension tables.
- The fact table contains quantitative data, such as sales amounts or quantities, and foreign keys linking to the dimension tables.
- Dimension tables are typically denormalized, meaning they contain all relevant attributes in a single table.

Example:

- Fact Table: Sales
  - Columns: **SalesID**, **ProductID**, **CustomerID**, **SaleAmount**, **Date**
- Dimension Tables:
  - Products: **ProductID**, **ProductName**, **Category**
  - Customers: **CustomerID**, **CustomerName**, **Region**
  - Date: **DateID**, **Date**, **Month**, **Year**

### **Snowflake Schema**

Overview:

- A snowflake schema is a more complex design that involves a central fact table surrounded by normalized dimension tables.
- Dimension tables may be split into additional related tables, reducing data redundancy.
- This structure often resembles a snowflake, with multiple layers of related tables.

Example:

- Fact Table: Sales
  - Columns: **SalesID**, **ProductID**, **CustomerID**, **SaleAmount**, **Date**

- Dimension Tables:
  - Products: ProductID, ProductName, CategoryID
  - Categories: CategoryID, CategoryName
  - Customers: CustomerID, CustomerName, RegionID
  - Regions: RegionID, RegionName
  - Date: DateID, Date, Month, Year

### **Slowly Changing Dimension:**

A Slowly Changing Dimension (SCD) manages changes to dimension data over time in a data warehouse:

- Type 0: No changes, data stays as is.
- Type 1: Overwrites old data with new values.
- Type 2: Keeps historical records by adding new rows for changes.
- Type 3: Adds new columns to track limited history.
- Type 4: Uses a separate historical table for old data.
- Type 6: Combines Type 1, 2, and 3 for hybrid tracking.

These types help manage how data changes are tracked based on the business need.

### **Junk Dimension:**

A Junk Dimension is a dimension table in a data warehouse that groups together low-cardinality, miscellaneous, or unrelated attributes (e.g., flags, indicators, or small descriptive data). Instead of creating separate dimensions for each attribute, these minor attributes are combined into a single junk dimension to simplify the schema and avoid creating multiple small, inefficient tables.

Example: If you have a set of flags like "IsActive," "IsPromotional," or "IsDiscounted," you could consolidate them into one junk dimension, reducing complexity in the fact table.

It's mainly used to optimize storage and improve query performance.

### **Degenerate Dimension:**

A Degenerate Dimension is a dimension in a data warehouse that doesn't have its own dimension table. Instead, it's represented by a column or set of columns directly in the fact table. Degenerate dimensions typically contain unique identifiers or transactional data like invoice numbers, order IDs, or receipt numbers, which are useful for analysis but don't have any associated attributes that would require a separate dimension table.

In short, degenerate dimensions are dimensions that exist within the fact table, without needing a separate dimension table for additional information.

## Dimensional Modeling Use Cases: From Normalized to Optimized

### Use Case 4: Telecom Network Performance

To model and optimize your telecom network performance data for efficient querying, we can follow these steps:

#### Step 1: Understand the Use Case and Schema Design

We need to analyze a large amount of data — 6 million daily network events — to monitor network performance, user behavior, and service quality. The normalized schema provided contains separate tables for users, towers, services, and network events.

#### Step 1: Set Up the Database

```
1 • CREATE DATABASE TelecomNetworkPerformance;
2 USE TelecomNetworkPerformance;
```

#### Step 2: Create the Star Schema

This structure optimizes the query performance for large datasets.

#### Dimension Tables

```
4 -- Dimension Tables
5 -- Creating dimension tables for Date, Tower, Service, and User.
6 • CREATE TABLE DimDate (
7     DateID INT PRIMARY KEY,
8     Date DATE,
9     Year INT,
10    Month INT,
11    Day INT,
12    Hour INT
13 );
14
15 • CREATE TABLE DimTower (
16     TowerID INT PRIMARY KEY,
17     Location VARCHAR(100),
18     Capacity INT
19 );
20
21 • CREATE TABLE DimService (
22     ServiceID INT PRIMARY KEY,
23     ServiceName VARCHAR(50),
24     ServiceType VARCHAR(50)
25 );
26
27 • CREATE TABLE DimUser (
28     UserID INT PRIMARY KEY,
29     Name VARCHAR(100),
30     PlantType VARCHAR(50)
31 );
```

## Fact Table

This table will store the 6 million network event records.

```
35      -- Fact Table
36      -- This table will store the 6 million network event records.
37 • CREATE TABLE FactNetworkEvents (
38     EventID INT PRIMARY KEY,
39     DateID INT,
40     TowerID INT,
41     ServiceID INT,
42     UserID INT,
43     Duration INT,
44     DataUsed INT,
45     FOREIGN KEY (DateID) REFERENCES DimDate(DateID),
46     FOREIGN KEY (TowerID) REFERENCES DimTower(TowerID),
47     FOREIGN KEY (ServiceID) REFERENCES DimService(ServiceID),
48     FOREIGN KEY (UserID) REFERENCES DimUser(UserID)
49 );
50
```

## Step 3: Populate Dimension Tables

Populate the dimension tables. This will remain a smaller dataset with limited records.

### 1. DimDate Table (7 Days of Hourly Data)

We generate hourly data for 7 days (168 records in total).

```

53      -- DimDate Table (7 Days of Hourly Data)
54      -- We generate hourly data for 7 days (168 records in total)
55  DELIMITER //
56
57 •   CREATE PROCEDURE InsertDimDateData()
58     BEGIN
59         DECLARE currentDate DATETIME DEFAULT '2024-10-01 00:00:00';
60         DECLARE endDate DATETIME DEFAULT '2024-10-08 00:00:00';
61         DECLARE dateID INT DEFAULT 1;
62
63     WHILE currentDate < endDate DO
64         INSERT INTO DimDate (DateID, Date, Year, Month, Day, Hour)
65             VALUES (dateID,
66                     CAST(currentDate AS DATE),
67                     YEAR(currentDate),
68                     MONTH(currentDate),
69                     DAY(currentDate),
70                     HOUR(currentDate));
71
72         -- Increment the currentDate by 1 hour
73         SET currentDate = DATE_ADD(currentDate, INTERVAL 1 HOUR);
74         SET dateID = dateID + 1;
75     END WHILE;
76 END //
77
78  DELIMITER ;
79 •   CALL InsertDimDateData();
80

```

## 2. DimTower Table

```

83      -- DimTower Table
84      -- Insert data for 2 telecom towers
85 •   INSERT INTO DimTower (TowerID, Location, Capacity)
86      VALUES (1, 'Tower A', 2000), (2, 'Tower B', 1500);
87
88

```

### 3. DimService Table

```
90      -- DimService Table
91      -- Insert data for 2 services (e.g., Voice Call and 4G Internet)
92 •    INSERT INTO DimService (ServiceID, ServiceName, ServiceType)
93      VALUES (1, 'Voice Call', 'Telephony'), (2, '4G Internet', 'Data');
94
```

### 4. DimUser Table

```
96
97      -- DimUser Table
98      -- Insert data for 2 users (sample data)
99 •    INSERT INTO DimUser (UserID, Name, PlanType)
100     VALUES (1, 'John Doe', 'Premium'), (2, 'Jane Doe', 'Standard');
101
```

## Step 4: Insert 6 Million Records into the Fact Table

Next, we insert **6 million records** directly into the `FactNetworkEvents` table.

```

104    -- Insert 6 Million Records into the Fact Table
105    -- Next, we insert 6 million records directly into the FactNetworkEvents table.
106    DELIMITER //
107
108 • CREATE PROCEDURE InsertFactNetworkEventsData()
109     BEGIN
110         DECLARE counter INT DEFAULT 1;
111         DECLARE randomDateID INT;
112         DECLARE randomTowerID INT;
113         DECLARE randomServiceID INT;
114         DECLARE randomUserID INT;
115         DECLARE randomDuration INT;
116         DECLARE randomDataUsed INT;
117
118     WHILE counter <= 6000000 DO
119         -- Generate random values for each column
120         SET randomDateID = FLOOR(RAND() * 168) + 1;    -- Random DateID (1 to 168)
121         SET randomTowerID = FLOOR(RAND() * 2) + 1;    -- Random TowerID (1 or 2)
122         SET randomServiceID = FLOOR(RAND() * 2) + 1;   -- Random ServiceID (1 or 2)
123         SET randomUserID = FLOOR(RAND() * 2) + 1;      -- Random UserID (1 or 2)
124         SET randomDuration = FLOOR(RAND() * 120);     -- Random Duration (0 to 120 minutes)
125         SET randomDataUsed = FLOOR(RAND() * 5000);    -- Random Data Used (0 to 5000 MB)
126
127         -- Insert the generated row into the table
128         INSERT INTO FactNetworkEvents (EventID, DateID, TowerID, ServiceID, UserID, Duration, DataUsed)
129             VALUES (counter, randomDateID, randomTowerID, randomServiceID, randomUserID, randomDuration, randomDataUsed);
130
131         -- Increment the counter
132         SET counter = counter + 1;
133     END WHILE;
134
135
136     DELIMITER ;
137 • CALL InsertFactNetworkEventsData();
138

```

This script will insert 6 million records by generating random values for **DateID**, **TowerID**, **ServiceID**, **UserID**, **Duration**, and **DataUsed**.

## Step 5: Optimize the Query

Now that the data is inserted, you can run an optimized query on the star schema. This query calculates the average duration and total data usage by location, service type, and hour.

```
142      -- Optimize the Query
143 •   SELECT dt.Location, ds.ServiceType, dd.Hour,
144           AVG(fne.Duration) AS AvgDuration, SUM(fne.DataUsed) AS TotalDataUsed
145     FROM FactNetworkEvents fne
146   JOIN DimDate dd ON fne.DateID = dd.DateID
147   JOIN DimTower dt ON fne.TowerID = dt.TowerID
148   JOIN DimService ds ON fne.ServiceID = ds.ServiceID
149 WHERE dd.Date >= DATEADD(day, -7, GETDATE())
150 GROUP BY dt.Location, ds.ServiceType, dd.Hour
151 ORDER BY dt.Location, ds.ServiceType, dd.Hour;
152
```

## Conclusion

The process involved building a star schema and inserting 6 million records directly into the fact table. After setting up the star schema, indexing, and running optimized queries, we should see improved performance on large datasets.

## Triggers in MySQL

```

1 •  create database trigger_practise;
2
3 •  use trigger_practise;
4
5 •  create table customers(id int auto_increment primary key,
6   name varchar(100),
7   email varchar(100));
8
9 •  create table email_changes_log(
10    id int auto_increment primary key,
11    customer_id int,
12    old_email varchar(100),
13    new_email varchar(100),
14    changed_at timestamp default current_timestamp);
15
17 •  insert into customers(name,email) values('Auahdahd','dqhduiqwh@gmail.com');
18
19 •  select * from customers;
20

```

Result Grid			Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
	id	name	email			
▶	1	Auahdahd	dqhduiqwh@gmail.com			
*	HULL	HULL	HULL			

```

21     DELIMITER //
22 •  CREATE TRIGGER log_email_changes
23   before update on customers
24   for each row
25   begin
26     if old.email!=new.email then
27       insert into email_changes_log(customer_id,old_email,new_email)
28         values(old.id,old.email, new.email);
29     end if;
30   end// 
31
32   delimiter ;
33
34 •  select * from email_changes_log;
35 •  update customers set email='jdiejweiiod@gmail.com' where id =1;
36 •  select * from customers;

```

Result Grid					
	id	customer_id	old_email	new_email	changed_at
▶	1	1	dqhduiqwh@gmail.com	jdiejweiiod@gmail.com	2024-10-11 14:59:38
*	NULL	NULL	NULL	NULL	NULL

```

21     DELIMITER //
22 •  CREATE TRIGGER log_email_changes
23   after update on customers
24   for each row
25   begin
26     if old.email!=new.email then
27       insert into email_changes_log(customer_id,old_email,new_email)
28         values(old.id,old.email, new.email);
29     end if;
30   end// 
31
32   delimiter ;
33
34 •  select * from email_changes_log;
35 •  update customers set email='jdiejweiiod@gmail.com' where id =1;
36 •  select * from customers;

```

Result Grid					
	id	customer_id	old_email	new_email	changed_at
▶	1	1	dqhduiqwh@gmail.com	jdiejweiiod@gmail.com	2024-10-11 14:59:38
*	NULL	NULL	NULL	NULL	NULL

# Day 20 - Concepts of Spring Boot

## Understanding Dependency Injection and Inversion of Control in Spring

### 1. Introduction to Dependency Injection (DI)

Dependency Injection is a design pattern that allows us to develop loosely coupled code. It's a technique for achieving Inversion of Control (IoC) between classes and their dependencies.

Step 1: Understanding the Problem DI Solves

Let's start with a simple example without DI:

```
java
public class TextEditor {
    private SpellChecker spellChecker;

    public TextEditor() {
        this.spellChecker = new SpellChecker();
    }
}
```

In this case, TextEditor is tightly coupled to SpellChecker. If we want to use a different spell checker, we'd have to modify the TextEditor class.

Step 2: Applying Dependency Injection

Now, let's refactor this using DI:

```
java
public class TextEditor {
    private SpellChecker spellChecker;
```

```
public TextEditor(SpellChecker spellChecker) {  
    this.spellChecker = spellChecker;  
}  
}
```

Now, TextEditor is not responsible for creating the SpellChecker. It's "injected" from outside.

## 2. Types of Dependency Injection

Spring supports three types of dependency injection. Let's explore each one.

Step 3: Constructor Injection

This is the most recommended form of DI in Spring.

```
java  
@Component  
public class TextEditor {  
    private final SpellChecker spellChecker;  
  
    @Autowired  
    public TextEditor(SpellChecker spellChecker) {  
        this.spellChecker = spellChecker;  
    }  
}
```

Spring will automatically inject a SpellChecker bean when creating a TextEditor.

Step 4: Setter Injection

Setter injection uses, well, setter methods:

```
java  
@Component
```

```
public class TextEditor {  
    private SpellChecker spellChecker;  
  
    @Autowired  
    public void setSpellChecker(SpellChecker spellChecker) {  
        this.spellChecker = spellChecker;  
    }  
}
```

### Step 5: Field Injection

Field injection injects dependencies directly into fields:

```
java  
@Component  
public class TextEditor {  
    @Autowired  
    private SpellChecker spellChecker;  
}
```

Note: While convenient, field injection is generally discouraged as it makes testing more difficult.

## 3. Inversion of Control (IoC)

IoC is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework.

### Step 6: Understanding IoC

In traditional programming, our custom code makes calls to a library. IoC is the exact opposite - the framework makes calls to our custom code.

### Step 7: Spring IoC Container

Spring's IoC container is responsible for managing object creation, configuring these objects, and managing their lifecycle.

There are two types of IoC containers in Spring:

1. BeanFactory
2. ApplicationContext (a more advanced container and extension of BeanFactory)

## Step 8: Configuring Spring IoC

We can configure the Spring IoC container using XML, Java annotations, or Java code. Let's look at an example using Java annotations:

```
java
@Configuration
public class AppConfig {
    @Bean
    public SpellChecker spellChecker() {
        return new SpellChecker();
    }

    @Bean
    public TextEditor textEditor(SpellChecker spellChecker) {
        return new TextEditor(spellChecker);
    }
}
```

This configuration tells Spring how to create and wire our objects.

## 4. Putting It All Together

Let's see how DI and IoC work together in a Spring application.

---

## Step 9: Creating the Application

```
java
@SpringBootApplication
public class TextEditorApplication {
    public static void main(String[] args) {
        ApplicationContext context = SpringApplication.run(TextEditorApplication.class, args);
        TextEditor editor = context.getBean(TextEditor.class);
        editor.spellCheck("Hello, wrld!");
    }
}
```

In this example:

1. Spring Boot sets up the ApplicationContext (IoC container).
2. The container creates and configures all the beans.
3. We retrieve the TextEditor bean from the container.
4. We use the TextEditor, which internally uses the SpellChecker that was injected.

## 5. Benefits of DI and IoC

- Reduced dependency between classes
- Easier unit testing through mocking
- Greater modularity
- Increased code reusability
- More flexible, configurable applications

By leveraging DI and IoC, Spring allows us to write more modular, testable, and maintainable code.

## Design Considerations: `@Autowired` vs `getBean()` in Spring

### 1. `@Autowired`

#### When to Use

In Spring-managed beans (e.g., `@Component`, `@Service`, `@Repository`, `@Controller`)

For dependencies that are required throughout the lifecycle of a bean

When you want to leverage Spring's dependency injection fully

#### Design Advantages

Loose Coupling: `@Autowired` promotes loose coupling as the dependent class doesn't need to know about the creation and management of its dependencies.

javaCopy@Service

```
public class OrderService {  
    private final PaymentService paymentService;
```

`@Autowired`

```
public OrderService(PaymentService paymentService) {  
    this.paymentService = paymentService;  
}  
}
```

Testability: It's easier to mock dependencies in unit tests.

javaCopy@Test

```
public void testOrderService() {  
    PaymentService mockPaymentService = mock(PaymentService.class);  
    OrderService orderService = new OrderService(mockPaymentService);  
    // ... test orderService  
}
```

Cleaner Code: Reduces boilerplate code for dependency lookup and instantiation.

Consistency: Ensures that the same instance of a dependency is used throughout the application (for singleton-scoped beans).

Lifecycle Management: Spring manages the complete lifecycle of autowired dependencies.

## 2. `getBean()`

When to Use

In classes not managed by Spring

When you need to choose between multiple implementations at runtime

For retrieving prototype-scoped beans multiple times

In factory patterns where bean selection happens dynamically

Design Advantages

Flexibility at Runtime: Allows for dynamic selection of dependencies.

`javaCopy@Component`

```
public class PaymentProcessor {  
    private final ApplicationContext context;
```

`@Autowired`

```
public PaymentProcessor(ApplicationContext context) {  
    this.context = context;  
}
```

```
public void processPayment(String paymentType) {  
    PaymentService paymentService = context.getBean(paymentType + "PaymentService",  
    PaymentService.class);  
    paymentService.process();  
}
```

Lazy Loading: Can delay bean creation until it's actually needed, potentially improving startup time.

`javaCopy@Component`

```
public class HeavyResourceUser {  
    private final ApplicationContext context;
```

`@Autowired`

```
public HeavyResourceUser(ApplicationContext context) {
```

```

        this.context = context;
    }

    public void useHeavyResource() {
        HeavyResource resource = context.getBean(HeavyResource.class);
        resource.performOperation();
    }
}

```

**Access to Multiple Implementations:** Useful when you have multiple implementations of an interface and need to choose at runtime.

```

javaCopy@Component
public class ReportGenerator {
    private final ApplicationContext context;

    @Autowired
    public ReportGenerator(ApplicationContext context) {
        this.context = context;
    }

    public void generateReport(String reportType) {
        ReportService reportService = context.getBean(reportType + "ReportService",
ReportService.class);
        reportService.generate();
    }
}

```

**Prototype Scope Handling:** When you need a new instance of a prototype-scoped bean each time.

```

javaCopy@Component
public class PrototypeManager {
    private final ApplicationContext context;

    @Autowired
    public PrototypeManager(ApplicationContext context) {
        this.context = context;
    }
}

```

```
}

public void doSomething() {
    PrototypeBean prototypeBean = context.getBean(PrototypeBean.class);
    prototypeBean.process();
}
}
```

### 3. Design Patterns and Use Cases

#### Dependency Injection Pattern

Prefer `@Autowired` for implementing the Dependency Injection pattern in Spring-managed beans.

It aligns well with the Inversion of Control principle.

#### Factory Pattern

Use `getBean()` when implementing a Factory pattern where the exact type of object is determined at runtime.

```
javaCopy@Component
public class AnimalFactory {
    private final ApplicationContext context;

    @Autowired
    public AnimalFactory(ApplicationContext context) {
        this.context = context;
    }

    public Animal createAnimal(String animalType) {
        return context.getBean(animalType, Animal.class);
    }
}
```

```

@Component
public class DependencyValidator {
    @Autowired(required = false)
    private OptionalService optionalService;
    @Autowired
    private RequiredService requiredService;

    @PostConstruct
    public void validateDependencies() {
        if (requiredService == null) {
            throw new IllegalStateException("RequiredService is not injected!");
        }
        if (optionalService == null) {
            System.out.println("OptionalService is not available, some features may be limited.");
        }
    }
}

```

## Key Differences from Constructor Initialization

- Timing: `@PostConstruct` methods are called after all dependency injection is complete, whereas constructors are called during bean instantiation.
- Access: `@PostConstruct` methods have full access to dependency-injected fields, which might not be initialized in the constructor.
- Inheritance: `@PostConstruct` methods in superclasses are called before those in subclasses.
- Exception Handling: `@PostConstruct` methods can throw checked exceptions, unlike constructors.

**A detailed, step-by-step example demonstrating the use of both `@Autowired` and `getBean()` in Spring. We'll create a simple application that manages a library system:**

### **Step 1: Set up the project**

1. Go to <https://start.spring.io/>
2. Choose:
  - Project: Maven
  - Language: Java
  - Spring Boot: 3.1.x
  - Group: com.example
  - Artifact: libraryapp
  - Packaging: Jar
  - Java: 17
3. Add Dependencies: Spring Web
4. Generate and download the project
5. Extract and open in your IDE

### **Step 2: Create the Book model**

Create a new package `com.example.libraryapp.model` and add a `Book` class:

```
java
package com.example.libraryapp.model;

public class Book {
    private String isbn;
    private String title;
    private String author;

    // Constructor, getters, and setters
}
```

### **Step 3: Create the LibraryService**

Create a package com.example.libraryapp.service and add:

```
java
package com.example.libraryapp.service;

import com.example.libraryapp.model.Book;
import org.springframework.stereotype.Service;
import java.util.HashMap;
import java.util.Map;

@Service
public class LibraryService {
    private Map<String, Book> books = new HashMap<>();

    public void addBook(Book book) {
        books.put(book.getIsbn(), book);
    }

    public Book getBook(String isbn) {
        return books.get(isbn);
    }
}
```

#### Step 4: Create BookController using @Autowired

Create a package com.example.libraryapp.controller and add:

```
java
package com.example.libraryapp.controller;

import com.example.libraryapp.model.Book;
import com.example.libraryapp.service.LibraryService;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.web.bind.annotation.*;  
  
 @RestController  
 @RequestMapping("/api/books")  
 public class BookController {  
  
     private final LibraryService libraryService;  
  
     @Autowired  
     public BookController(LibraryService libraryService) {  
         this.libraryService = libraryService;  
     }  
  
     @PostMapping  
     public void addBook(@RequestBody Book book) {  
         libraryService.addBook(book);  
     }  
  
     @GetMapping("/{isbn}")  
     public Book getBook(@PathVariable String isbn) {  
         return libraryService.getBook(isbn);  
     }  
 }
```

## Step 5: Create BeanRetrievalService using getBean()

Add a new service:

```
java  
package com.example.libraryapp.service;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.ApplicationContext;  
import org.springframework.stereotype.Service;
```

```
@Service
public class BeanRetrievalService {

    private final ApplicationContext context;

    @Autowired
    public BeanRetrievalService(ApplicationContext context) {
        this.context = context;
    }

    public <T> T getBean(Class<T> beanClass) {
        return context.getBean(beanClass);
    }
}
```

## Step 6: Create AdminController using getBean()

Add another controller:

```
java
package com.example.libraryapp.controller;

import com.example.libraryapp.model.Book;
import com.example.libraryapp.service.BeanRetrievalService;
import com.example.libraryapp.service.LibraryService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/admin")
public class AdminController {

    private final BeanRetrievalService beanRetrievalService;
```

```
@Autowired
public AdminController(BeanRetrievalService beanRetrievalService) {
    this.beanRetrievalService = beanRetrievalService;
}

@PostMapping("/books")
public void addBook(@RequestBody Book book) {
    LibraryService libraryService = beanRetrievalService.getBean(LibraryService.class);
    libraryService.addBook(book);
}

@GetMapping("/books/{isbn}")
public Book getBook(@PathVariable String isbn) {
    LibraryService libraryService = beanRetrievalService.getBean(LibraryService.class);
    return libraryService.getBook(isbn);
}
```

## Step 7: Run and Test

1. Run the main application class (LibraryappApplication.java)
2. Use Postman or curl to test:

Add a book (using `@Autowired`):

POST http://localhost:8080/api/books

Content-Type: application/json

```
{
  "isbn": "1234567890",
  "title": "Spring in Action",
  "author": "Craig Walls"
}
```

Get a book (using `@Autowired`):

GET `http://localhost:8080/api/books/1234567890`

Add a book (using `getBean()`):

POST `http://localhost:8080/api/admin/books`

`Content-Type: application/json`

```
{  
    "isbn": "0987654321",  
    "title": "Pro Spring 5",  
    "author": "Iuliana Cosmina"  
}
```

Get a book (using `getBean()`):

GET `http://localhost:8080/api/admin/books/0987654321`

This example demonstrates:

1. `@Autowired`: Used in `BookController` for dependency injection of `LibraryService`.
2. `getBean()`: Used in `AdminController` to retrieve `LibraryService` at runtime.

The `@Autowired` approach is simpler and more common, while `getBean()` offers more flexibility at runtime but is generally used less frequently.

```
src > main > java > com > example > bookapi > model > Book.java > {} com.example.bookapi.model
1 package com.example.bookapi.model;
2
3 public class Book {
4     private String id;
5     private String title;
6     private String author;
7     private int year;
8
9     public Book(String id, String title, String author, int year){
10         this.title=title;
11         this.author=author;
12         this.year= year;
13         this.id=id;
14     }
15     //getters and setters
16     public String getId() {
17         return id;
18     }
19     /**
20      * @param id
21      */
22     public void setId(String id){
23         this.id=id;
24     }
25     public String getTitle(){
26         return title;
27     }
28     public void setTitle(String title){
29         this.title=title;
30     }
31     public String getAuthor(){
32         return author;
33     }
34     public void setAuthor(String author){
35         this.author=author;
36     }
37     public int getYear(){
38         return year;
39     }
40     public void setYear(int year){
41         this.year = year;
42     }
43 }
```

```
src > main > java > com > example > librarayapp > controller > BookController.java > {} com.example.librarayapp.controller
 1 package com.example.librarayapp.controller;
 2 import org.springframework.beans.factory.annotation.Autowired;
 3 import org.springframework.web.bind.annotation.*;
 4 import com.example.bookapi.model.*;
 5 import com.example.librarayapp.service.LibraryService;
 6 @RestController
 7 @RequestMapping("/api/books")
 8 public class BookController {
 9
10     private final LibraryService libraryService;
11
12     @Autowired
13     public BookController(LibraryService libraryService){
14         this.libraryService = libraryService;
15     }
16
17     @PostMapping
18     public void addBook(@RequestBody Book book){
19         libraryService.addBook(book);
20     }
21
22     @GetMapping("/{isbn}")
23     public Book getBook(@PathVariable String isbn){
24         return libraryService.getBook(isbn);
25     }
26 }
27
```

```
src > main > java > com > example > librarayapp > service > BeanRetrievalService.java > {} com.example.librarayapp.service
 1 package com.example.librarayapp.service;
 2
 3 import org.springframework.context.ApplicationContext;
 4 import org.springframework.stereotype.Service;
 5 import org.springframework.beans.factory.annotation.Autowired;
 6 @Service
 7 public class BeanRetrievalService {
 8     private final ApplicationContext context;
 9
10     @Autowired
11     public BeanRetrievalService(ApplicationContext context) {
12         this.context = context;
13     }
14     public <T> T getBean(Class<T> beanClass){
15         return context.getBean(beanClass);
16     }
17 }
18 }
```

```
src > main > java > com > example > librarayapp > LibraryApplication.java > ...
1 package com.example.librarayapp;
2 import org.springframework.boot.SpringApplication;
3 import org.springframework.boot.autoconfigure.SpringBootApplication;
4
5 @SpringBootApplication
6 public class LibraryApplication {
7     Run | Debug
8     public static void main(String[] args) {
9         SpringApplication.run(LibraryApplication.class, args);
10    }
11 }
```

Ctrl+L to chat, Ctrl+K to generate

```
PS C:\Codebase\demo\demo> mvn clean install
[INFO] Scanning for projects...
[INFO] 
[INFO] < com.example:demo >
[INFO] Building demo 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] [ jar ]
[INFO]
[INFO] --- clean:3.3.2:clean (default-clean) @ demo ---
[INFO] Deleting C:\Codebase\demo\demo\target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ demo ---
[INFO] Copying 1 resource from src\main\resources to target\classes
[INFO] Copying 0 resource from src\main\resources to target\classes
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ demo ---
[INFO] Recompiling the module because of changed source code.
[INFO] Compiling 6 source files with javac [debug parameters release 22] to target\classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ demo ---
[INFO] skip non existing resourceDirectory C:\Codebase\demo\demo\src\test\resources
[INFO]
[INFO] --- compiler:3.13.0:testCompile (default-testCompile) @ demo ---
[INFO] Recompiling the module because of changed dependency.
[INFO]
[INFO] --- surefire:3.2.5:test (default-test) @ demo ---
[INFO]
[INFO] --- jar:3.4.2:jar (default-jar) @ demo ---
[INFO] Building jar: C:\Codebase\demo\demo\target\demo-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot:3.3.4:repackage (repackage) @ demo ---
[INFO] Replacing main artifact C:\Codebase\demo\demo\target\demo-0.0.1-SNAPSHOT.jar with repackaged archive, adding nested dependencies in BOOT-INF/.
[INFO] The original artifact has been renamed to C:\Codebase\demo\demo\target\demo-0.0.1-SNAPSHOT.jar.original
[INFO]
[INFO] --- install:3.1.3:install (default-install) @ demo ---
[INFO] Installing C:\Codebase\demo\demo\pom.xml to C:\Users\Anjali\.m2\repository\com\example\demo\0.0.1-SNAPSHOT\demo-0.0.1-SNAPSHOT.pom
[INFO] Installing C:\Codebase\demo\demo\target\demo-0.0.1-SNAPSHOT.jar to C:\Users\Anjali\.m2\repository\com\example\demo\0.0.1-SNAPSHOT\demo-0.0.1-SNAPSHOT.jar
[INFO] 
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 10.006 s
[INFO] Finished at: 2024-10-14T21:44:23+05:30
[INFO]
```

```

PS C:\Codebase\demo\demo> mvn spring-boot:run
[INFO] Scanning for projects...
[INFO] [INFO] ------------------------------------------------------------------------
[INFO] [INFO] Building demo 0.0.1-SNAPSHOT
[INFO] [INFO]   from pom.xml
[INFO] [INFO] [INFO] [jar] ...
[INFO] [INFO] [INFO] --- spring-boot:3.3.4:run (default-cli) > test-compile @ demo ---
[INFO] [INFO] [INFO] --- resources:3.1.3:resources (default-resources) @ demo ---
[INFO] [INFO] [INFO] Copying 1 resource from src\main\resources to target\classes
[INFO] [INFO] [INFO] Copying 0 resource from src\main\resources to target\classes
[INFO] [INFO] [INFO] --- compiler:3.13.0:compile (default-compile) @ demo ---
[INFO] [INFO] [INFO] Nothing to compile - all classes are up to date.
[INFO] [INFO] [INFO] --- resources:3.1.3:listResources (default-testResources) @ demo ---
[INFO] [INFO] [INFO] skip non existing resourceDirectory C:\Codebase\demo\demo\target\resources
[INFO] [INFO] [INFO] --- compiler:3.13.0:testCompile (default-testCompile) @ demo ---
[INFO] [INFO] [INFO] Nothing to compile - all classes are up to date.
[INFO] [INFO] [INFO] <>> spring-boot:3.3.4:run (default-cli) < test-compile @ demo <<<
[INFO] [INFO] [INFO] --- spring-boot:3.3.4:run (default-cli) @ demo ---
[INFO] [INFO] [INFO] Attaching agents: []
[INFO] [INFO] [INFO] 
[INFO] [INFO] [INFO] 
[INFO] [INFO] [INFO] :: Spring Boot ::      (v3.3.4)
[INFO] [INFO] [INFO] 2024-10-14T21:55:11.503+05:30  INFO 7232 --- [demo] [           main] c.e.libraryapp.libraryApplication : Starting LibraryApplication using Java 22.0.1 with PID 7232: C:\Codebase\demo\demo\target\classes started by Anjali in C:\Codebase\demo\demo
[INFO] [INFO] [INFO] 2024-10-14T21:55:11.504+05:30  INFO 7232 --- [demo] [           main] c.e.libraryapp.libraryApplication : No active profile set, falling back to 1 default profile: "default"
[INFO] [INFO] [INFO] 2024-10-14T21:55:11.505+05:30  INFO 7232 --- [demo] [           main] o.a.c.c.C.[Tomcat].[localhost].[/] : StandardEngine [Tomcat/10.1.30]
[INFO] [INFO] [INFO] 2024-10-14T21:55:11.464+05:30  INFO 7232 --- [demo] [           main] o.apache.catalina.core.StandardService : Starting Service [Tomcat]
[INFO] [INFO] [INFO] 2024-10-14T21:55:11.465+05:30  INFO 7232 --- [demo] [           main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.30]
[INFO] [INFO] [INFO] 2024-10-14T21:55:11.616+05:30  INFO 7232 --- [demo] [           main] w.s.c.ServletWebServerApplicationContext : Initializing Spring embedded WebApplicationContext
[INFO] [INFO] [INFO] 2024-10-14T21:55:11.618+05:30  INFO 7232 --- [demo] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2084 ms
[INFO] [INFO] [INFO] 2024-10-14T21:55:14.534+05:30  INFO 7232 --- [demo] [           main] o.s.w.e.embedded.tomcat.TomcatWebServer : Tomcat started on port 8088 (http) with context path '/'
[INFO] [INFO] [INFO] 2024-10-14T21:55:14.534+05:30  INFO 7232 --- [demo] [           main] c.e.libraryapp.libraryApplication : Started LibraryApplication in 4.075 seconds (process running for 4.878)

```

HTTP <http://localhost:8080/api/books>

POST <http://localhost:8080/api/books>

Params Authorization Headers (9) **Body** Scripts Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON**

```

1  {
2    "isbn": "1234567890",
3    "title": "Spring in Action",
4    "author": "Craig Walls"
5  }
6

```

Body Cookies Headers (4) Test Results 200 OK

HTTP <http://localhost:8080/api/books/1234567890>

GET <http://localhost:8080/api/books/1234567890>

Params Authorization Headers (9) **Body** Scripts Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON**

```

1  {
2    "isbn": "1234567890",
3    "title": "Spring in Action",
4    "author": "Craig Walls"
5  }
6

```

Body Cookies Headers (4) Test Results 200 OK

The screenshot shows a REST client interface with two requests:

**POST** <http://localhost:8080/api/admin/books>

Body (raw JSON):

```
1 {
2   "isbn": "0987654321",
3   "title": "Pro Spring 5",
4   "author": "Iuliana Cosmina"
5 }
```

**GET** <http://localhost:8080/api/admin/books/0987654321>

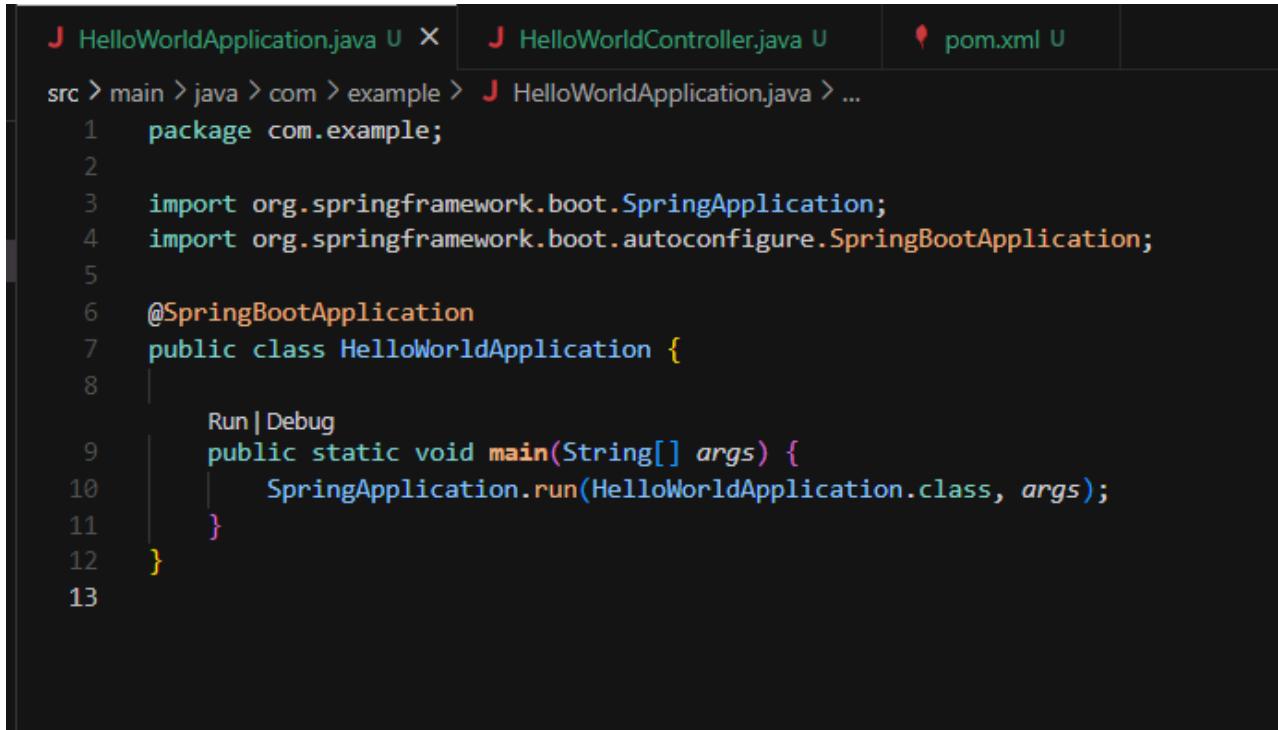
Body (raw JSON):

```
1 {
2   "isbn": "0987654321",
3   "title": "Pro Spring 5",
4   "author": "Iuliana Cosmina"
5 }
```

Both requests resulted in a **200 OK** response.

## Day 21 - Spring Boot Applications

### HelloWorld Application



The screenshot shows a code editor with three tabs at the top: 'HelloWorldApplication.java' (selected), 'HelloWorldController.java', and 'pom.xml'. The 'HelloWorldApplication.java' tab contains the following Java code:

```
src > main > java > com > example > J HelloWorldApplication.java > ...
1 package com.example;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class HelloWorldApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(HelloWorldApplication.class, args);
11     }
12 }
13
```

```
src > main > java > com > example > J HelloWorldController.java > ...
1 package com.example;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 public class HelloWorldController {
8
9     @GetMapping("/api/hello")
10    public String helloWorld() {
11        return "Hello, World!";
12    }
13
14
15}
```

HTTP <http://localhost:8080/api/hello>

---

GET [▼](#) http://localhost:8080/api/hello

Params Authorization Headers (7) Body Scripts Settings

Query Params

	Key	Value
	Key	Value

---

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text [▼](#)

```
1 Hello, World!
```

```

src > main > java > com > example > WeatherController.java > getWeatherWithCountry(String, String)
1 package com.example;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.PathVariable;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @RestController
8 public class WeatherController {
9
10     @GetMapping("/api/weather/{city}")
11     public String getWeather(@PathVariable String city) {
12         return "The weather in " + city + " is nice today.";
13     }
14
15     @GetMapping("/api/weather/{city}/{country}")
16     public String getWeatherWithCountry(@PathVariable String city, @PathVariable String country) {
17         return "The weather in " + city + ", " + country + " is 25 degrees Celsius today.";
18     }
19 }
20

```

HTTP <http://localhost:8080/api/weather/London>

GET <http://localhost:8080/api/weather/London>

Params Authorization Headers (7) Body Scripts Settings

**Query Params**

	Key	Value	Descr
	Key	Value	Descr

Body Cookies Headers (5) Test Results 200 OK

Pretty Raw Preview Visualize Text ▾

1 Weather information not available for London

```
src > main > java > com > example > J WeatherApiApplication.java > ...
1 package com.example;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class WeatherApiApplication {
8     Run | Debug
9     public static void main(String[] args) {
10         SpringApplication.run(WeatherApiApplication.class, args);
11     }
12
13 }
```

```
src > main > java > com > example > J WeatherController.java > WeatherController > getWeather(String)
1 package com.example;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.*;
5
6 @RestController
7 @RequestMapping("/api")
8 public class WeatherController {
9
10     private final WeatherService weatherService;
11
12     @Autowired
13     public WeatherController(WeatherService weatherService) {
14         this.weatherService = weatherService;
15     }
16
17     @GetMapping("/weather/{city}")
18     public String getWeather(@PathVariable String city) {
19         return weatherService.getWeather(city);
20     }
21
22     @PostMapping("/weather/{city}")
23     public String updateWeather(@PathVariable String city, @RequestParam String condition) {
24         weatherService.updateWeather(city, condition);
25         return "Weather updated for " + city;
26     }
27 }
28 }
```

```
src > main > java > com > example > WeatherService.java > WeatherService > updateWeather(String, String)
1 package com.example;
2
3 import org.springframework.stereotype.Service;
4 import java.util.HashMap;
5 import java.util.Map;
6
7 @Service
8 public class WeatherService {
9
10     private Map<String, String> weatherData = new HashMap<>();
11
12     public String getWeather(String city) {
13         return weatherData.getOrDefault(city, "Weather information not available for " + city);
14     }
15
16     public void updateWeather(String city, String condition) {
17         weatherData.put(city, condition);
18     }
19 }
```

The screenshot shows a REST client interface with the following details:

- Overview:** Overview | GET http://localhost:8080/api/... | +
- Request:** **HTTP** http://localhost:8080/api/weather/London
- Method:** GET
- URL:** http://localhost:8080/api/weather/London
- Params:** Params (selected), Authorization, Headers (7), Body, Scripts, Settings
- Query Params:** A table with columns Key, Value, and Description. It contains one row with Key: Value and Description: Descript.
- Body:** Body, Cookies, Headers (5), Test Results
- Test Results:** 200 OK
- Content:** 1 Weather information not available for London

## Weather Update and Retrieval Service

### **Flowchart of the Application:**

#### 1. Client Interaction:

- The process starts with a client sending an HTTP request to our Weather API.

#### 2. WeatherController:

- The request is received by the WeatherController.
- It has two main endpoints:
  - a. GET /api/weather/{city}: Maps to getWeather method
  - b. POST /api/weather/{city}: Maps to updateWeather method

#### 3. Dependency Injection:

- WeatherService is injected into WeatherController.
- WeatherConfig is injected into WeatherService.

#### 4. GET Request Flow:

- When a GET request is received, WeatherController.getWeather calls WeatherService.getWeather.
- WeatherService.getWeather checks if the city exists in the weatherData map.
- If the city exists, it returns the stored weather condition.
- If the city doesn't exist, it returns the default condition from WeatherConfig.

#### 5. POST Request Flow:

- When a POST request is received, WeatherController.updateWeather calls WeatherService.updateWeather.
- WeatherService.updateWeather updates the weatherData map with the new weather condition for the specified city.

#### 6. WeatherConfig:

- WeatherConfig is configured by application.properties.
- It provides the defaultCondition to WeatherService.

#### 7. In-Memory Data Storage:

- The weatherData map in WeatherService acts as an in-memory database, storing the weather conditions for different cities.

This flow chart now accurately represents the structure and flow of our Java code:

- It shows the correct method calls between WeatherController and WeatherService.
- It accurately represents how WeatherService uses the weatherData map and WeatherConfig.
- It illustrates the configuration flow from application.properties to WeatherConfig.

This representation should help students understand:

1. The role of each component (Controller, Service, Config) in the application.
2. How HTTP requests are handled and processed through the application layers.
3. The use of dependency injection to connect components.
4. How data is stored and retrieved using the in-memory weatherData map.
5. The role of external configuration via application.properties.

```
src > main > java > com > example > J WeatherApiApplication.java > ...  
  
1 package com.example;  
2  
3 import org.springframework.boot.SpringApplication;  
4 import org.springframework.boot.autoconfigure.SpringBootApplication;  
5  
6 @SpringBootApplication  
7 public class WeatherApiApplication {  
8     Run | Debug  
9     public static void main(String[] args) {  
10         SpringApplication.run(WeatherApiApplication.class, args);  
11     }  
12 }  
13 }
```

```
src > main > java > com > example > J WeatherConfig.java > ...
1 package com.example;
2
3 import org.springframework.boot.context.properties.ConfigurationProperties;
4 import org.springframework.context.annotation.Configuration;
5
6 @Configuration
7 @ConfigurationProperties(prefix = "weather")
8 public class WeatherConfig {
9     private String defaultCondition;
10
11     public String getDefaultCondition() {
12         return defaultCondition;
13     }
14
15     public void setDefaultCondition(String defaultCondition) {
16         this.defaultCondition = defaultCondition;
17     }
18 }
19
20
```

```
src > main > java > com > example > J WeatherController.java > ...
1 package com.example;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.*;
5 import javax.validation.constraints.NotBlank;
6 import org.springframework.validation.annotation.Validated;
7
8 @RestController
9 @RequestMapping("/api")
10 @Validated
11 public class WeatherController {
12
13     private final WeatherService weatherService;
14
15     @Autowired
16     public WeatherController(WeatherService weatherService) {
17         this.weatherService = weatherService;
18     }
19
20     @GetMapping("/weather/{city}")
21     public String getWeather(@PathVariable @NotBlank String city) {
22         return weatherService.getWeather(city);
23     }
24
25     @PostMapping("/weather/{city}")
26     public String updateWeather(@PathVariable @NotBlank String city,
27                                 @RequestParam @NotBlank String condition) {
28         weatherService.updateWeather(city, condition);
29         return "Weather updated for " + city;
30     }
31 }
32
```

```
src > main > java > com > example > J WeatherService.java > WeatherService > updateWeather(String, String)
```

```
1 package com.example;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import java.util.HashMap;
6 import java.util.Map;
7
8 @Service
9 public class WeatherService{
10     private final WeatherConfig weatherConfig;
11     private Map<String, String> weatherData = new HashMap<>();
12     @Autowired
13     public WeatherService(WeatherConfig weatherConfig){
14         this.weatherConfig = weatherConfig;
15     }
16     public String getWeather(String city){
17         return weatherData.getOrDefault(city, weatherConfig.getDefaultCondition());
18     }
19     public void updateWeather(String city, String condition){
20         weatherData.put(city, condition);
21     }
22 }
```

The screenshot shows a REST client interface with the following details:

- HTTP:** http://localhost:8080/api/weather/London
- Method:** GET
- URL:** http://localhost:8080/api/weather/London
- Params:** (None)
- Headers:** (5)
- Body:** (Empty)
- Cookies:** (Empty)
- Headers:** (5)
- Test Results:** (Empty)
- Status:** 200 OK
- Content:** 1 Sunny

HTTP <http://localhost:8080/api/weather/London?condition=rainy>

POST <http://localhost:8080/api/weather/London?condition=rainy>

Params • Authorization Headers (8) Body Scripts Settings

Query Params

Key	Value	Description
condition	rainy	
Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK

Pretty Raw Preview Visualize Text

1 Weather updated for London

## Day 22 - SQL Views and Spring Boot Applications

### SQL Views:

```
CREATE VIEW view_name AS  
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

Example

Let's create a view based on an example employees table:

```
1 •  create database view_test;  
2  
3 •  use view_test;  
4  
5 •  CREATE TABLE employees (  
6      id INT PRIMARY KEY,  
7      first_name VARCHAR(50),  
8      last_name VARCHAR(50),  
9      department VARCHAR(50),  
10     salary DECIMAL(10, 2)  
11 );  
12  
13     -- Insert some sample data  
14 •  INSERT INTO employees VALUES  
15     (1, 'John', 'Doe', 'IT', 75000),  
16     (2, 'Jane', 'Smith', 'HR', 65000),  
17     (3, 'Mike', 'Johnson', 'IT', 80000);  
18  
19 •  create view it_employee as  
20     select id, first_name, last_name, salary  
21     from employees  
22     where department = 'IT';
```

```
24 • select * from it_employee;
```

	id	first_name	last_name	salary
▶	1	John	Doe	75000.00
	3	Mike	Johnson	80000.00

```
26 • show create view it_employee;
```

	View	Create View	character_set_client	collation_connection
▶	it_employee	CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` VIEW `it_employee` AS select `id`, `first_name`, `last_name`, `salary` from `employees`	utf8mb4	utf8mb4_0900_ai_ci

```
28 • explain select * from it_employee;
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	employees	NULL	ALL	NULL	NULL	NULL	NULL	3	33.33	Using where

```
30 • select view_definition  
      from information_schema.views  
      where table_schema = 'view_test'  
      and table_name = 'it_employee';  
31  
32  
33  
34  
35  
36
```

	VIEW_DEFINITION
▶	select `view_test`.`employees`.`id` AS `id`, `...

## Procedure: Refresh Materialized View for IT Employees

```
36    DELIMITER //
37 •  CREATE PROCEDURE refresh_mv_it_employee()
38  BEGIN
39      INSERT INTO mv_it_employee
40          SELECT id, first_name, last_name, department, salary
41          FROM employees
42          WHERE department = 'IT';
43  END //
44 //
45
46 •  CREATE EVENT refresh_mv_it_employee_daily
47     ON SCHEDULE EVERY 1 DAY
48     STARTS CURRENT_DATE + INTERVAL 1 DAY
49     DO
50         CALL refresh_mv_it_employee();
51
52     -- Calling the correct procedure and selecting from the correct table
53
54     CALL refresh_mv_it_employee();
55     SELECT * FROM mv_it_employee;
56
```

Result Grid					
	id	first_name	last_name	department	salary
	1	John	Doe	IT	75000.00
▶	3	Mike	Johnson	IT	80000.00

## Checking Running Queries in MySQL

In MySQL, there isn't a direct equivalent to Oracle's v\$session table. However, MySQL provides other ways to check currently running queries and session information.

### ## Methods to Check Running Queries in MySQL

#### 1. \*Using SHOW PROCESSLIST Command\*

The most common way to see what queries are currently running is to use the SHOW PROCESSLIST command:

```
sql
```

```
SHOW PROCESSLIST;
```

This command shows you a list of current server threads, including information about each connection and the query it's executing (if any).

#### 2. \*Querying the INFORMATION\_SCHEMA.PROCESSLIST Table\*

For more flexibility in filtering and formatting results, you can query the PROCESSLIST table:

```
sql
```

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST;
```

This allows you to use WHERE clauses and joins for more specific information.

#### 3. \*Using the performance\_schema Database\*

In MySQL 5.7 and later, you can use tables in the performance\_schema database for more detailed information:

```
sql
```

```
SELECT * FROM performance_schema.threads
```

---

```
JOIN performance_schema.events_statements_current USING (thread_id);
```

This joins thread information with current statement events, giving you a comprehensive view of running queries.

#### 4. \*MySQL Workbench\*

If you're using MySQL Workbench, you can use the "Client Connections" section under "PERFORMANCE" to view current connections and their queries graphically.

## Example: Filtering for Specific Queries

If you want to find specific types of queries, you can add WHERE clauses. For instance, to find long-running queries:

sql

```
SELECT id, user, host, db, command, time, state, info  
FROM INFORMATION_SCHEMA.PROCESSLIST  
WHERE command != 'Sleep'  
AND time > 5;
```

This will show queries that have been running for more than 5 seconds.

#### ## Notes

- The visibility of queries depends on your user privileges. You might need PROCESS privilege to see all queries.
- The 'info' column in PROCESSLIST contains the actual SQL of the running query, but it might be truncated for very long queries.
- For security reasons, some installations might limit the information available about other users' queries.

## Continuing previous weather example:

### Refactoring WeatherService to Use WeatherRecord:

```
src > main > java > com > example > WeatherRecord.java > WeatherRecord
1 package com.example;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class WeatherRecord {
7     private String city;
8     private String condition;
9
10    public WeatherRecord(){
11
12    }
13
14    public void initialize(String city, String condition){
15        this.city = city;
16        this.condition = condition;
17    }
18
19    public String getCity(){
20        return city;
21    }
22
23    public String getCondition(){
24        return condition;
25    }
26    public void setCity(String city){
27        this.city = city;
28    }
29    public void setCondition(String condition){
30        this.condition = condition;
31    }
32 }
```

```
src > main > java > com > example > WeatherService.java > WeatherService
 1 package com.example;
 2
 3 import java.util.HashMap;
 4 import java.util.Map;
 5 import org.springframework.beans.factory.annotation.Autowired;
 6 import org.springframework.stereotype.Service;
 7
 8 @Service
 9 public class WeatherService {
10     private final WeatherConfig weatherConfig;
11
12     private final WeatherRecord weatherRecord;
13
14     @Autowired
15     public WeatherService(WeatherConfig weatherConfig, WeatherRecord weatherRecord){
16         this.weatherConfig = weatherConfig;
17         this.weatherRecord = weatherRecord;
18     }
19
20     private Map<String, WeatherRecord> weatherData = new HashMap<>();
21
22     public String getWeather(String city){
23         return weatherData.get(city).getCondition();
24     }
25     public void updateWeatherData(String city, String weather){
26
27         if(weather!=null){
28             weatherRecord.initialize(city, weather);
29             weatherData.put(city, weatherRecord);
30         }else{
31             weatherRecord.initialize(city, weatherConfig.getDefaultCondition());
32             weatherData.put(city, weatherRecord);
33         }
34     }
35 }
36 }
```

**http://localhost:8080/api/weather/London**

**GET** http://localhost:8080/api/weather/London

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK

Pretty Raw Preview Visualize Text

```
1 rainy
```

---

**http://localhost:8080/api/weather/London?condition=sunny**

**POST** http://localhost:8080/api/weather/London?condition=sunny

Params Authorization Headers (8) Body Scripts Settings

Query Params

Key	Value	Description
condition	sunny	
Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK

Pretty Raw Preview Visualize Text

```
1 Weather updated for London
```

---

**http://localhost:8080/api/weather/London**

**GET** http://localhost:8080/api/weather/London

Params Authorization Headers (7) Body Scripts Settings

Query Params

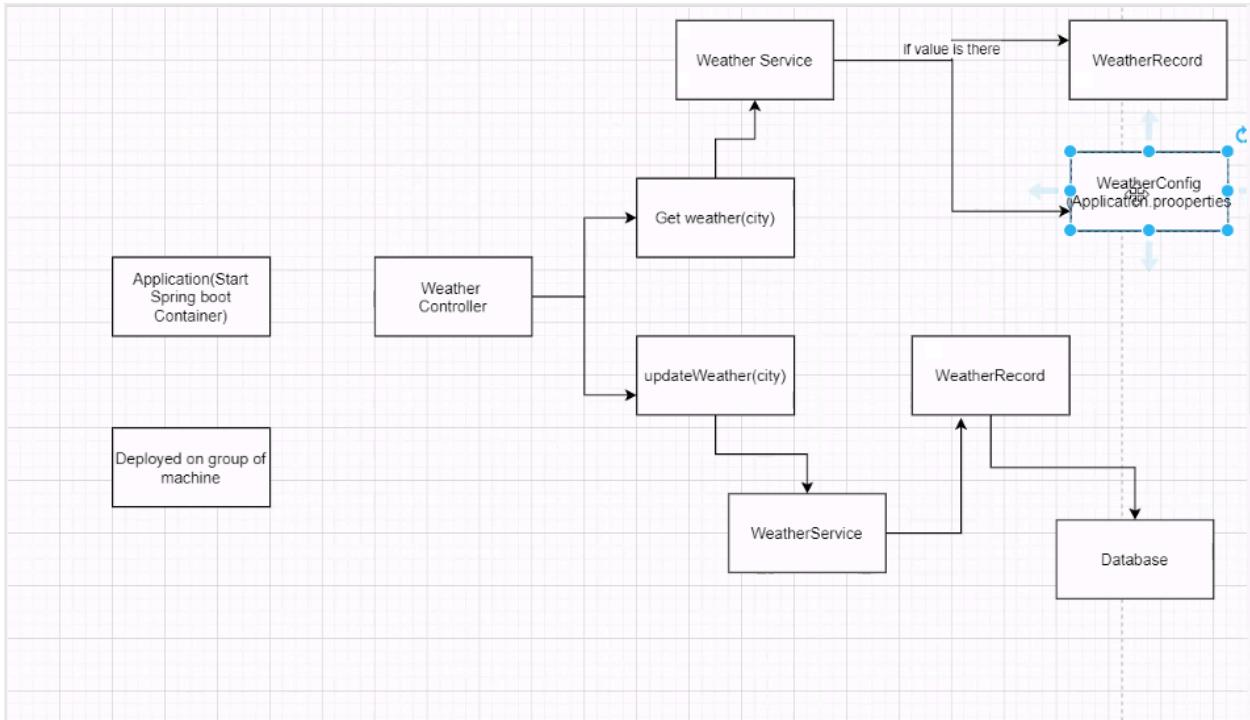
Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK

Pretty Raw Preview Visualize Text

```
1 sunny
```

## Flowchart Explanation:



## Day 23 - Spring Boot Continues...

### Spring Boot Data JPA:

Spring Data JPA simplifies database interactions in Spring Boot applications by offering an abstraction over JPA (Java Persistence API). It enables developers to work with relational databases using Java objects (entities) without writing boilerplate code.

Key Features of Spring Data JPA:

1. **Repositories:** Interface-driven repository abstraction for CRUD operations, with methods such as `save()`, `findById()`, `findAll()`, and `delete()`.
2. **JPQL (Java Persistence Query Language):** Query entities using JPQL, which is similar to SQL but operates on entity objects rather than tables.
3. **Derived Query Methods:** Create query methods by following naming conventions (e.g., `findByName(String name)` will generate a query to find entities by the "name" field).
4. **Custom Queries:** You can define custom queries using the `@Query` annotation for complex logic.
5. **Pagination and Sorting:** Built-in support for pagination and sorting results with minimal effort.

Example:

Entity Class:

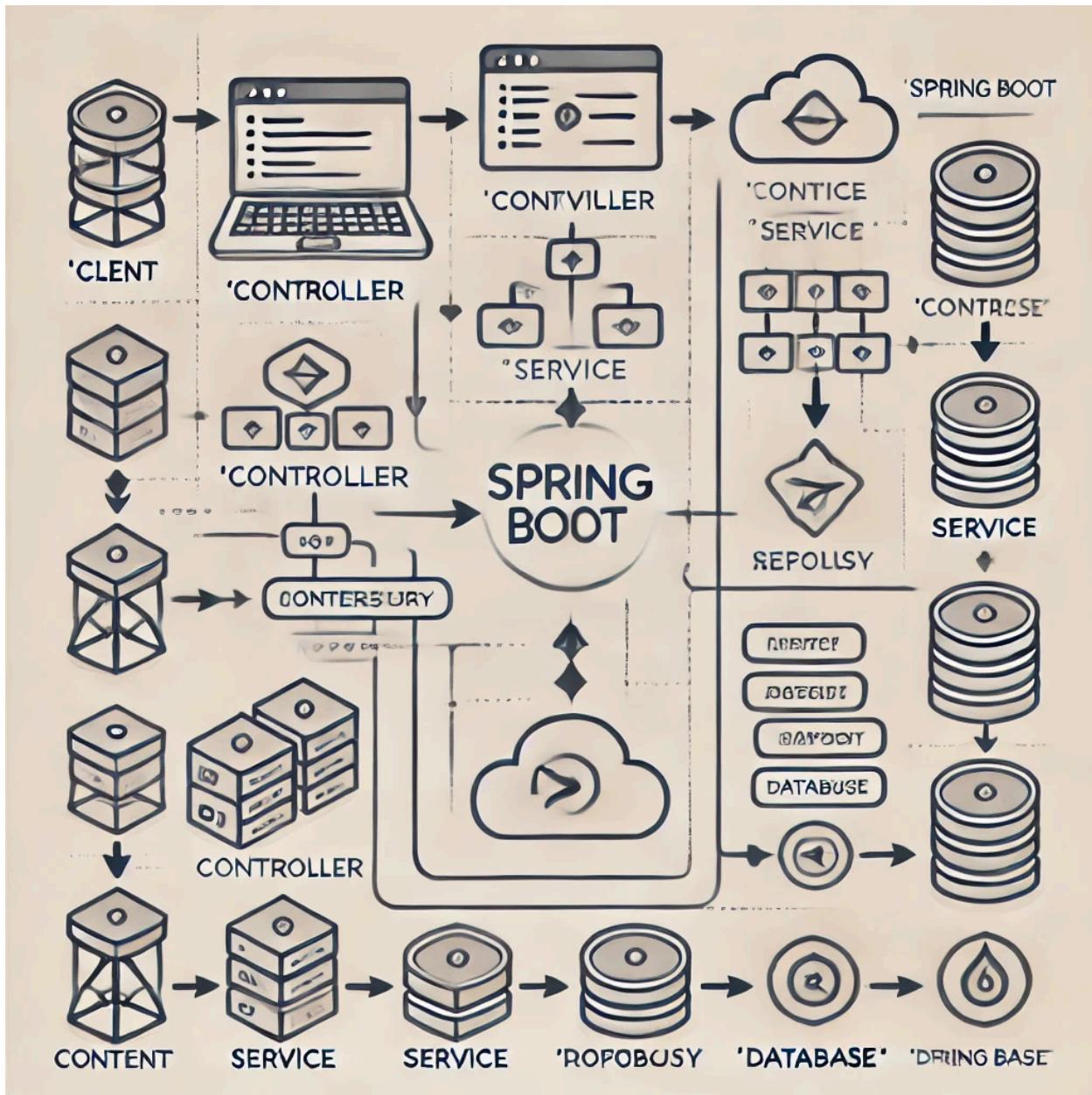
```
@Entity  
public class User {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
    private String name;  
    private String email;  
    // getters and setters  
}
```

Repository Interface:

```
public interface UserRepository extends JpaRepository<User, Long> {  
    // Derived query method  
    List<User> findByName(String name);  
}
```

Service Layer:

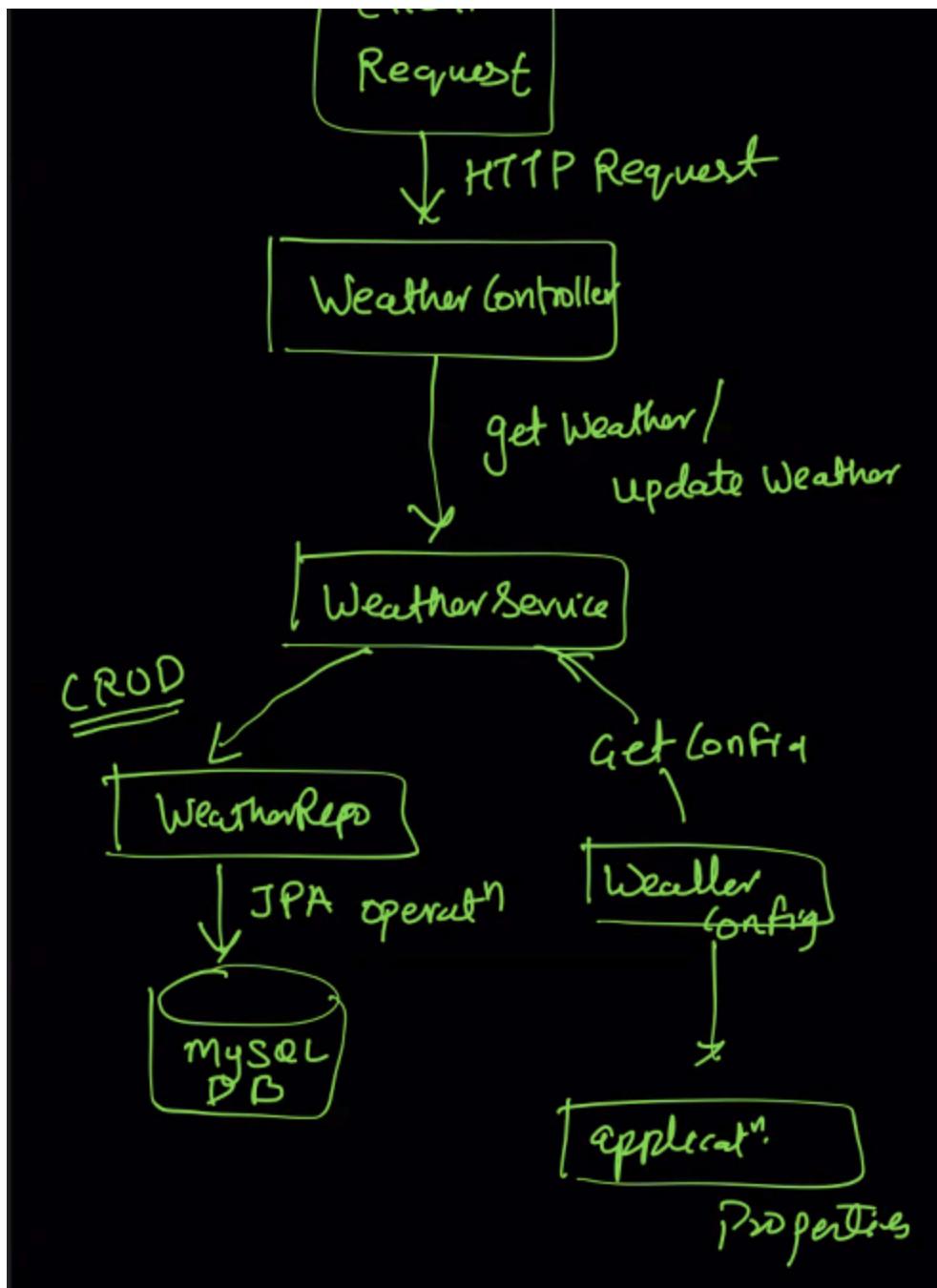
```
@Service  
public class UserService {  
    @Autowired  
    private UserRepository userRepository;  
  
    public List<User> getUsersByName(String name) {  
        return userRepository.findByName(name);  
    }  
}
```



Flow:

- The client sends an HTTP request to the controller.
- The controller calls the service layer, which processes the data or logic.
- The service interacts with the repository, which retrieves or saves entities from/to the database.
- The HTTP response is returned to the client after processing.

## Spring Boot Weather Application:



This diagram outlines the flow of a Spring Boot Weather Application using Spring Data JPA:

1. Client Request: The client sends an HTTP request to the application to get or update weather information.

- 
- 2. Weather Controller: Handles the incoming HTTP request, processes it, and calls the Weather Service to handle the business logic (e.g., fetching or updating weather data).
  - 3. Weather Service: This is the core of the application where the business logic happens. It performs two main actions:
    - Communicates with the WeatherRepo (Weather Repository) to perform CRUD operations (using JPA) on the MySQL Database.
    - Retrieves configuration from WeatherConfig for various settings, which may come from `application.properties` or other configuration files.
  - 4. WeatherRepo: This is the repository layer that interacts with the MySQL database, leveraging Spring Data JPA to perform database operations such as fetching or updating weather data.
  - 5. WeatherConfig: Holds configuration settings, likely sourced from external configuration files like `application.properties` or environment variables, which provide specific parameters to the service.

## Weather Application using Database Connectivity:

```
demo > src > main > java > com > example > WeatherConfig.java
 1 package com.example;
 2 import java.util.HashMap;
 3 import java.util.Map;
 4
 5 import org.springframework.boot.context.properties.ConfigurationProperties;
 6 import org.springframework.context.annotation.Configuration;
 7 @Configuration
 8 @ConfigurationProperties(prefix = "weather")
 9 public class WeatherConfig {
10     private String defaultCondition;
11     private Map<String, TemperatureRange> cityTempratureRanges = new HashMap<>();
12
13     public String getDefaultCondition(){
14         return defaultCondition;
15     }
16     public void setDefaultCondition(String defaultCondition){
17         this.defaultCondition = defaultCondition;
18     }
19     public Map<String, TemperatureRange> getCityTempratureRanges(){
20         return this.cityTempratureRanges;
21     }
22     public void setCityTempratureRanges(Map<String, TemperatureRange> cityTempratureRanges){
23         this.cityTempratureRanges = cityTempratureRanges;
24     }
25     public static class TemperatureRange{
26         private double min;
27         private double max;
28         public double getMin(){
29             return min;
30         }
31         public void setMin(double min){
32             this.min = min;
33         }
34         public double getMax(){
35             return max;
36         }
37         public void setMax(double max){
38             this.max = max;
39         }
40     }
41 }
```

```

demo > src > main > java > com > example > WeatherController.java > {} com.example
  1 package com.example;
  2
  3 import org.springframework.web.bind.annotation.RestController;
  4
  5 import java.util.Map;
  6 import org.springframework.web.bind.annotation.GetMapping;
  7 import org.springframework.web.bind.annotation.PathVariable;
  8 import org.springframework.web.bind.annotation.RequestMapping;
  9 import org.springframework.web.bind.annotation.RequestParam;
 10 import org.springframework.web.bind.annotation.PostMapping;
 11 import org.springframework.web.bind.annotation.DeleteMapping;
 12
 13 import javax.validation.constraints.NotBlank;
 14 import org.springframework.beans.factory.annotation.Autowired;
 15 import org.springframework.validation.annotation.Validated;
 16 import java.util.List;
 17 import org.springframework.http.ResponseEntity;
 18 import org.springframework.web.bind.annotation.PutMapping;
 19 import org.springframework.web.bind.annotation.RequestBody;
 20 import org.springframework.http.HttpStatus;
 21 @RestController
 22 @RequestMapping("/api/weather")
 23 @Validated
 24 public class WeatherController {
 25
 26     private final WeatherService weatherService;
 27     @Autowired
 28     public WeatherController(WeatherService weatherService){
 29         this.weatherService = weatherService;
 30     }
 31
 32     @GetMapping("/all")
 33     public List<WeatherRecord> getAllWeather(){
 34         return weatherService.getAllWeather();
 35     }
 36
 37     @GetMapping("/{city}")
 38     public ResponseEntity<WeatherRecord> getWeatherByCity(@PathVariable @NotBlank String city) {
 39         WeatherRecord weather = weatherService.getWeatherByCity(city);
 40         return weather != null ? ResponseEntity.ok(weather) : ResponseEntity.notFound().build();
 41     }
 42
 43     @PostMapping("/create")
 44     public WeatherRecord addWeather(@RequestBody WeatherRecord weatherRecord){
 45         return weatherService.addWeather(weatherRecord);
 46     }
 47
 48     @PutMapping("/{id}")
 49     public ResponseEntity<WeatherRecord> updateWeather(@PathVariable Long id, @RequestBody WeatherRecord weatherRecord){
 50         if(!weatherService.getAllWeather().stream().anyMatch(weather -> weather.getId().equals(id))){
 51             return ResponseEntity.notFound().build();
 52         }
 53         weatherRecord.setId(id);
 54         return ResponseEntity.ok(weatherService.updateWeather(weatherRecord));
 55     }
 56     @DeleteMapping("/{id}")
 57     public ResponseEntity<Void> deleteWeather(@PathVariable Long id){
 58         if(!weatherService.getAllWeather().stream().anyMatch(weather -> weather.getId().equals(id))){
 59             return ResponseEntity.notFound().build();
 60         }
 61         weatherService.deleteWeather(id);
 62         return ResponseEntity.noContent().build();
 63     }
 64
 65 }
 66

```

```
demo > src > main > java > com > example > WeatherRecord.java > {} com.example
 1 package com.example;
 2
 3 import jakarta.persistence.*;
 4
 5 @Entity
 6 @Table(name="weather_records")
 7 public class WeatherRecord {
 8     @Id
 9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10     private Long id;
11     @Column(nullable = false,unique = true)
12     private String city;
13     @Column(nullable = false)
14     private int temperature;
15     @Column(nullable = false)
16     private String weatherCondition;
17
18     public WeatherRecord(){}
19
20 }
21
22     public void initialize(String city, int temperature){
23         this.city = city;
24         this.temperature = temperature;
25     }
26
27     public Long getId(){
28         return id;
29     }
30     public void setId(Long id){
31         this.id = id;
32     }
33
34     public String getCity(){
35         return city;
36     }
37
38     public String getWeatherCondition(){
39         return weatherCondition;
40     }
41     public void setCity(String city){
42         this.city = city;
43     }
44     public void setWeatherCondition(String weatherCondition){
45         this.weatherCondition = weatherCondition;
46     }
47     public void setTemperature(int temperature){
48         this.temperature = temperature;
49     }
50     public int getTemperature(){
51         return temperature;
52     }
53 }
```

```
demo > src > main > java > com > example > J WeatherRepository.java > {} com.example
1 package com.example;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5 import java.util.Optional;
6 @Repository
7 public interface WeatherRepository extends JpaRepository<WeatherRecord, Long> {
8
9     Optional<WeatherRecord> findByCity(String city);
10 }
11
```

```
demo > src > main > java > com > example > J WeatherService.java > {} com.example
1 package com.example;
2
3 import java.util.HashMap;
4 import java.util.Map;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7 import java.util.List;
8 @Service
9 public class WeatherService {
10     private final WeatherRepository weatherRepository;
11     @Autowired
12     public WeatherService(WeatherRepository weatherRepository){
13         this.weatherRepository = weatherRepository;
14     }
15
16     public List<WeatherRecord> getAllWeather(){
17         return weatherRepository.findAll();
18     }
19
20     public WeatherRecord getWeatherByCity(String city){
21         return weatherRepository.findByCity(city).orElseThrow(() -> new RuntimeException(message:"City not found"));
22     }
23
24     public WeatherRecord addWeather(WeatherRecord weatherRecord){
25         return weatherRepository.save(weatherRecord);
26     }
27     public void deleteWeather(Long id){
28         weatherRepository.deleteById(id);
29     }
30     public WeatherRecord updateWeather(WeatherRecord weatherRecord){
31         return weatherRepository.save(weatherRecord);
32     }
33
34 }
```

```
application.properties U ● J WeatherConfig.java U J WeatherRecord.java U J WeatherRepository.java U J HelloController.java U J HelloService.java U J HelloServiceImp.java U J HelloControllerImp.java U J HelloControllerTest.java U J HelloServiceTest.java U J HelloServiceImpTest.java U J HelloControllerImpTest.java U
```

demo > src > main > resources > application.properties

```
1 spring.application.name=demo
2 server.port=8081
3 logging.level.org.springframework.web=DEBUG
4 logging.level.org.apache.tomcat=DEBUG
5 logging.level.org.apache.catalina=DEBUG
6 logging.level.org.apache.coyote=DEBUG
7 logging.level.org.apache.tomcat.util.net=DEBUG
8 logging.level.org.apache.tomcat.util.threads=DEBUG
9 logging.level.org.apache.tomcat.util.buf=DEBUG
10 logging.level.org.apache.tomcat.util.http=DEBUG
11 logging.level.org.apache.tomcat.util.res=DEBUG
12 logging.level.org.apache.tomcat.util.scan=DEBUG
13 logging.level.org.apache.tomcat.util.scan=DEBUG
14 weather.default-condition=Moderate
15 weather.city-temprature-ranges.NewYork.min=10
16 weather.city-temprature-ranges.NewYork.max=20
17 weather.city-temprature-ranges.London.min=10
18 weather.city-temprature-ranges.London.max=20
19 weather.city-temprature-ranges.Paris.min=10
20 weather.city-temprature-ranges.Paris.max=20
21 weather.city-temprature-ranges.Tokyo.min=10
22 weather.city-temprature-ranges.Tokyo.max=20
23
24
25 #Add database configuration
26 spring.datasource.url=jdbc:mysql://localhost:3306/weatherdb?useSSL=false&serverTimezone=UTC
27 spring.datasource.username=root
28 spring.datasource.password=root
29 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
30 spring.jpa.hibernate.ddl-auto=update
31 spring.jpa.show-sql=true
32 spring.jpa.properties.hibernate.format_sql=true
33 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
34
```

```

demo > # pom.xml
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 3   xsi:schemaLocation="http://maven.apache.org/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
 4     <modelVersion>4.0.0</modelVersion>
 5     <parent>
 6       <groupId>org.springframework.boot</groupId>
 7       <artifactId>spring-boot-starter-parent</artifactId>
 8       <version>3.3.4</version>
 9       <relativePath/> <!-- Lookup parent from repository -->
10     </parent>
11     <groupId>com.example</groupId>
12     <artifactId>demo</artifactId>
13     <version>0.0.1-SNAPSHOT</version>
14     <name>demo</name>
15     <description>Demo project for Spring Boot</description>
16     <url/>
17     <licenses>
18       <license/>
19     </licenses>
20     <developers>
21       <developer/>
22     </developers>
23     <scm>
24       <connection/>
25       <developerConnection/>
26       <tag/>
27       <url/>
28     </scm>
29     <properties>
30       <java.version>21</java.version>
31     </properties>
32     <dependencies>
33       <dependency>
34         <groupId>org.springframework.boot</groupId>
35         <artifactId>spring-boot-starter</artifactId>
36       </dependency>
37       <dependency>
38         <groupId>org.springframework.boot</groupId>
39         <artifactId>spring-boot-starter-web</artifactId>
40       </dependency>
41       <dependency>
42         <groupId>org.springframework</groupId>
43         <artifactId>spring-beans</artifactId>
44       </dependency>
45       <dependency>
46         <groupId>org.springframework</groupId>
47         <artifactId>spring-web</artifactId>
48       </dependency>
49       <dependency>
50
51         <groupId>org.springframework.boot</groupId>
52         <artifactId>spring-boot-starter-test</artifactId>
53         <scope>test</scope>
54       </dependency>
55       <dependency>
56         <groupId>org.springframework.boot</groupId>
57         <artifactId>spring-boot-starter-validation</artifactId>
58       </dependency>
59       <dependency>
60         <groupId>javax.validation</groupId>
61         <artifactId>validation-api</artifactId>
62         <version>2.0.0.Final</version>
63       </dependency>
64       <dependency>
65         <groupId>org.springframework.boot</groupId>
66         <artifactId>spring-boot-starter-data-jpa</artifactId>
67       </dependency>
68       <dependency>
69         <groupId>mysql</groupId>
70         <artifactId>mysql-connector-java</artifactId>
71         <version>8.0.33</version>
72       </dependency>
73     </dependencies>
74
75     <build>
76       <plugins>
77         <plugin>
78           <groupId>org.springframework.boot</groupId>
79           <artifactId>spring-boot-maven-plugin</artifactId>
80           <configuration>
81             <mainClass>com.example.HelloWorldApplication</mainClass>
82           </configuration>
83         </plugin>
84       </plugins>
85     </build>
86
87   </project>
88

```

**HTTP** <http://localhost:8081/api/weather/create>

**POST** <http://localhost:8081/api/weather/create>

Params Authorization Headers (9) **Body** Scripts Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON**

```

1 {
2   "id":1,
3   "city":"NewYork",
4   "temperaturize":25,
5   "weatherCondition":"sunny"
6 }
```

Body Cookies Headers (5) Test Results **200 OK**

Pretty Raw Preview Visualize **JSON**

```

1 {
2   "id": 1,
3   "city": "NewYork",
4   "temperaturize": 25,
5   "weatherCondition": "sunny"
6 }
```

**HTTP** <http://localhost:8081/api/weather/NewYork>

**GET** <http://localhost:8081/api/weather/NewYork>

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results **200 OK**

Pretty Raw Preview Visualize **JSON**

```

1 {
2   "id": 1,
3   "city": "NewYork",
4   "temperature": 25,
5   "weatherCondition": "sunny"
6 }
```

```
1 •  create database weatherdb;
2 •  use weatherdb;
3 •  show tables;
4
5 •  select * from weather_records
6
```

Result Grid				
	id	city	temperature	weather_condition
▶	1	NewYork	25	sunny
*	NULL	NULL	NULL	NULL

## Day 24 - SQL Practice Questions

### Problem Statement:

You're working with a company that manages projects across different departments. They want to analyze project performance, employee contributions, and department efficiency. Your task is to write a SQL query that:

```
1 •  create database projects;
2 •  use projects;
3
4 •  CREATE TABLE Departments (
5      DepartmentID INT PRIMARY KEY,
6      DepartmentName VARCHAR(50)
7 );
8
9 •  INSERT INTO Departments (DepartmentID, DepartmentName)
10    VALUES
11      (1, 'Engineering'),
12      (2, 'Marketing'),
13      (3, 'Sales');
14
15
16 •  CREATE TABLE Employees (
17      EmployeeID INT PRIMARY KEY,
18      EmployeeName VARCHAR(50),
19      DepartmentID INT,
20      FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
21 );
22
23 •  INSERT INTO Employees (EmployeeID, EmployeeName, DepartmentID)
24    VALUES
25      (1, 'John Doe', 1),
26      (2, 'Jane Smith', 1),
27      (3, 'Emily Johnson', 2),
28      (4, 'Michael Brown', 2),
29      (5, 'Sarah Davis', 3);
```

```

27      (3, 'Emily Johnson', 2),
28      (4, 'Michael Brown', 2),
29      (5, 'Sarah Davis', 3);
30
31
32 • CREATE TABLE Projects (
33     ProjectID INT PRIMARY KEY,
34     ProjectName VARCHAR(100),
35     DepartmentID INT,
36     StartDate DATE,
37     EndDate DATE,
38     EmployeeID INT,
39     FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID),
40     FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
41 );
42
43 • INSERT INTO Projects (ProjectID, ProjectName, DepartmentID, StartDate, EndDate, EmployeeID)
44 VALUES
45     (1, 'Engineering Project A', 1, '2024-01-01', '2024-02-01', 1),
46     (2, 'Engineering Project B', 1, '2024-02-15', '2024-03-15', 2),
47     (3, 'Engineering Project C', 1, '2024-04-01', '2024-05-01', 1),
48     (4, 'Marketing Campaign X', 2, '2024-01-10', '2024-02-10', 3),
49     (5, 'Marketing Campaign Y', 2, '2024-03-01', '2024-04-01', 4),
50     (6, 'Sales Initiative 1', 3, '2024-01-15', '2024-02-15', 5),
51     (7, 'Sales Initiative 2', 3, '2024-03-01', '2024-03-30', 5),
52     (8, 'Marketing Campaign Z', 2, '2024-05-01', '2024-06-01', 3);
53
54
55
56 -- 1. Rank employees within each department based on the number of projects they've completed:
57
58 • WITH EmployeeProjectCount AS (
59     SELECT
60         e.EmployeeID,
61         e.EmployeeName,
62         e.DepartmentID,
63         COUNT(p.ProjectID) AS ProjectCount
64     FROM
65         Employees e
66     JOIN
67         Projects p ON e.EmployeeID = p.EmployeeID
68     GROUP BY
69         e.EmployeeID, e.EmployeeName, e.DepartmentID
70 )
71     SELECT
72         epc.EmployeeID,
73         epc.EmployeeName,
74         epc.DepartmentID,
75         epc.ProjectCount,
76         ROW_NUMBER() OVER (PARTITION BY epc.DepartmentID ORDER BY epc.ProjectCount DESC) AS RankInDepartment
77     FROM
78         EmployeeProjectCount epc;
79

```

Result Grid | Filter Rows:  Export: Wrap Cell Content:

	EmployeeID	EmployeeName	DepartmentID	ProjectCount	RankInDepartment
▶	1	John Doe	1	2	1
	2	Jane Smith	1	1	2
	3	Emily Johnson	2	2	1
	4	Michael Brown	2	1	2
	5	Sarah Davis	3	2	1

```
81      -- 2. Calculate the average project duration for each department:  
82  
83 •      SELECT  
84          DepartmentID,  
85          AVG(DATEDIFF(EndDate, StartDate)) AS AvgProjectDuration  
86      FROM  
87          Projects  
88      GROUP BY  
89          DepartmentID;  
90  
91
```

---

Result Grid | Filter Rows:  Export: Wrap Cell Content:

	DepartmentID	AvgProjectDuration
▶	1	30.0000
	2	31.0000
	3	30.0000

```

91
92    -- 3. Identify projects that took longer than the department's average:
93
94 • WITH AvgDuration AS (
95     SELECT
96         DepartmentID,
97         AVG(DATEDIFF(EndDate, StartDate)) AS AvgProjectDuration
98     FROM
99        Projects
100    GROUP BY
101        DepartmentID
102    )
103    SELECT
104        p.ProjectID,
105        p.ProjectName,
106        p.DepartmentID,
107        DATEDIFF(p.EndDate, p.StartDate) AS ProjectDuration,
108        ad.AvgProjectDuration
109    FROM
110        Projects p
111    JOIN
112        AvgDuration ad ON p.DepartmentID = ad.DepartmentID
113    WHERE
114        DATEDIFF(p.EndDate, p.StartDate) > ad.AvgProjectDuration;
115

```

		Result Grid	Filter Rows:	Export:	Wrap Cell Content:
	ProjectID	ProjectName	DepartmentID	ProjectDuration	AvgProjectDuration
▶	1	Engineering Project A	1	31	30.0000
	6	Sales Initiative 1	3	31	30.0000

```

-- 4. Find the top 3 most efficient employees in each department (based on average project duration):
WITH EmployeeEfficiency AS (
    SELECT
        e.EmployeeID,
        e.EmployeeName,
        e.DepartmentID,
        AVG(DATEDIFF(p.EndDate, p.StartDate)) AS AvgProjectDuration
    FROM
        Employees e
    JOIN
        Projects p ON e.EmployeeID = p.EmployeeID
    GROUP BY
        e.EmployeeID, e.EmployeeName, e.DepartmentID
),
RankedEmployees AS (
    SELECT
        ee.EmployeeID,
        ee.EmployeeName,
        ee.DepartmentID,
        ee.AvgProjectDuration,
        ROW_NUMBER() OVER (PARTITION BY ee.DepartmentID ORDER BY ee.AvgProjectDuration ASC) AS RankInDepartment
    FROM
        EmployeeEfficiency ee
)
SELECT
    re.EmployeeID,
    re.EmployeeName,
    re.DepartmentID,
    re.AvgProjectDuration
FROM
    RankedEmployees re
WHERE
    re.RankInDepartment <= 3;

```

Result Grid | Filter Rows:  Export: Wrap Cell Content:

	EmployeeID	EmployeeName	DepartmentID	AvgProjectDuration
▶	2	Jane Smith	1	29.0000
	1	John Doe	1	30.5000
	3	Emily Johnson	2	31.0000
	4	Michael Brown	2	31.0000
	5	Sarah Davis	3	30.0000

```

153      -- 5. Compare each project's duration to the previous project in the same department:
154
155 • WITH ProjectDurations AS (
156     SELECT
157         p.ProjectID,
158         p.ProjectName,
159         p.DepartmentID,
160         DATEDIFF(p.EndDate, p.StartDate) AS ProjectDuration,
161         LAG(DATEDIFF(p.EndDate, p.StartDate)) OVER (PARTITION BY p.DepartmentID ORDER BY p.StartDate) AS PreviousProjectDuration
162     FROM
163         Projects p
164     )
165     SELECT
166         pd.ProjectID,
167         pd.ProjectName,
168         pd.DepartmentID,
169         pd.ProjectDuration,
170         pd.PreviousProjectDuration
171     FROM
172         ProjectDurations pd;

```

Result Grid					
	ProjectID	ProjectName	DepartmentID	ProjectDuration	PreviousProjectDuration
▶	1	Engineering Project A	1	31	NULL
	2	Engineering Project B	1	29	31
	3	Engineering Project C	1	30	29
	4	Marketing Campaign X	2	31	NULL
	5	Marketing Campaign Y	2	31	31
	8	Marketing Campaign Z	2	31	31
	6	Sales Initiative 1	3	31	NULL
	7	Sales Initiative 2	3	29	31

## Flow Chart:

[Input Tables]

```

|-- Projects (ProjectID, ProjectName, DepartmentID, StartDate, EndDate, EmployeeID)
|-- Employees (EmployeeID, EmployeeName, DepartmentID)
|-- Departments (DepartmentID, DepartmentName)
|
```

V

[ProjectStats CTE]

```

|-- JOIN: Projects, Employees, Departments
|-- Calculate:
|   - ProjectDuration (DATEDIFF)
|   - ProjectCount (COUNT(*) OVER)
|   - AvgDeptDuration (AVG OVER)
|-- Group By: None (Window functions used)
|
```

v

[RankedEmployees CTE]

|

-- Input: ProjectStats CTE

-- Calculate:

  - EmployeeRank (RANK() OVER)

  - EfficiencyRank (DENSE\_RANK() OVER)

-- Partition By: DepartmentID

-- Order By: ProjectCount DESC, ProjectDuration ASC

|

v

[ProjectComparison CTE]

|

-- Input: Projects table

-- Calculate:

  - PrevProjectDuration (LAG())

  - DurationDifference

-- Partition By: DepartmentID

-- Order By: StartDate

|

v

[Final SELECT]

|

-- JOIN: RankedEmployees, ProjectStats, ProjectComparison

-- Calculate: DurationComparison (CASE statement)

-- Filter: EfficiencyRank <= 3

-- Order By: DepartmentName, EfficiencyRank, ProjectDuration

|

v

[Output]

DepartmentName, EmployeeName, ProjectCount, EmployeeRank,  
EfficiencyRank, ProjectName, ProjectDuration, AvgDeptDuration,  
DurationComparison, PrevProjectDuration, DurationDifference

## Cumulative Sum and Moving Average:

```
177 • CREATE TABLE Sales (
178     SaleDate DATE,
179     Amount DECIMAL(10, 2)
180 );
181
182 •     INSERT INTO Sales (SaleDate, Amount) VALUES
183     ('2023-01-01', 100),
184     ('2023-01-02', 150),
185     ('2023-01-03', 200),
186     ('2023-01-04', 120),
187     ('2023-01-05', 180);
188
189
190     -- Cumulative Sum:
191
192 •     SELECT
193         SaleDate,
194         Amount,
195         SUM(Amount) OVER (ORDER BY SaleDate) AS CumulativeSum
196     FROM Sales;
197
198
199
200
```

Result Grid			
	SaleDate	Amount	CumulativeSum
▶	2023-01-01	100.00	100.00
	2023-01-02	150.00	250.00
	2023-01-03	200.00	450.00
	2023-01-04	120.00	570.00
	2023-01-05	180.00	750.00

```
199      -- Moving Average:  
200  
201 •  SELECT  
202      SaleDate,  
203      Amount,  
204      AVG(Amount) OVER (ORDER BY SaleDate ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS MovingAverage  
205  FROM Sales;  
206
```

Result Grid			
	SaleDate	Amount	MovingAverage
▶	2023-01-01	100.00	100.000000
	2023-01-02	150.00	125.000000
	2023-01-03	200.00	150.000000
	2023-01-04	120.00	156.666667
	2023-01-05	180.00	166.666667

## Problem Statement:

You're working for an e-commerce company that wants to analyze its sales data. The company has three main tables: Customers, Orders, and Products. Your task is to write a SQL query that provides the following insights:

```
213 • Ⓜ CREATE TABLE Customers (
214     CustomerID INT PRIMARY KEY,
215     Name VARCHAR(100),
216     Email VARCHAR(100),
217     RegistrationDate DATE,
218     Segment VARCHAR(20)
219 );
220
221 • Ⓜ CREATE TABLE Products (
222     ProductID INT PRIMARY KEY,
223     Name VARCHAR(100),
224     Category VARCHAR(50),
225     Price DECIMAL(10, 2)
226 );
227
228 • Ⓜ CREATE TABLE Orders (
229     OrderID INT PRIMARY KEY,
230     CustomerID INT,
231     OrderDate DATE,
232     TotalAmount DECIMAL(10, 2),
233     FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
234 );
235
236 • Ⓜ CREATE TABLE OrderDetails (
237     OrderID INT,
238     ProductID INT,
239     Quantity INT,
240     FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
241     FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
242 );
```

```
246      -- Insert sample data into Customers
247  •  INSERT INTO Customers (CustomerID, Name, Email, RegistrationDate, Segment)
248      VALUES
249          (1, 'John Doe', 'john@example.com', '2022-01-15', 'Regular'),
250          (2, 'Jane Smith', 'jane@example.com', '2022-02-20', 'Premium'),
251          (3, 'Bob Johnson', 'bob@example.com', '2022-03-10', 'Regular'),
252          (4, 'Alice Brown', 'alice@example.com', '2022-04-05', 'Premium'),
253          (5, 'Charlie Davis', 'charlie@example.com', '2022-05-01', 'Regular');
254
255      -- Insert sample data into Products
256  •  INSERT INTO Products (ProductID, Name, Category, Price)
257      VALUES
258          (1, 'Laptop', 'Electronics', 999.99),
259          (2, 'Smartphone', 'Electronics', 599.99),
260          (3, 'T-shirt', 'Clothing', 19.99),
261          (4, 'Jeans', 'Clothing', 49.99),
262          (5, 'Book', 'Books', 14.99);
263
264      -- Insert sample data into Orders
265  •  INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount)
266      VALUES
267          (1, 1, '2023-01-10', 1019.98),
268          (2, 2, '2023-02-15', 649.98),
269          (3, 3, '2023-03-20', 34.98),
270          (4, 4, '2023-04-25', 1049.98),
271          (5, 5, '2023-05-30', 64.98),
272          (6, 1, '2023-06-05', 614.98),
273          (7, 2, '2023-07-10', 1014.98),
274          (8, 3, '2023-08-15', 599.99),
275          (9, 4, '2023-09-20', 69.98),
276          (10, 5, '2023-10-25', 999.99);
```

```
275     (9, 4, '2023-09-20', 69.98),  
276     (10, 5, '2023-10-25', 999.99);  
277  
278     -- Insert sample data into OrderDetails  
279 • INSERT INTO OrderDetails (OrderID, ProductID, Quantity)  
280     VALUES  
281     (1, 1, 1), (1, 3, 1),  
282     (2, 2, 1), (2, 4, 1),  
283     (3, 3, 1), (3, 5, 1),  
284     (4, 1, 1), (4, 4, 1),  
285     (5, 3, 1), (5, 5, 1),  
286     (6, 2, 1), (6, 3, 1),  
287     (7, 1, 1), (7, 5, 1),  
288     (8, 2, 1),  
289     (9, 3, 1), (9, 4, 1),  
290     (10, 1, 1);
```

```
293      -- 1. Top 5 customers by total spending
294 •  SELECT
295      C.CustomerID,
296      C.Name,
297      SUM(O.TotalAmount) AS TotalSpending
298  FROM
299      Customers C
300  JOIN
301      Orders O ON C.CustomerID = O.CustomerID
302  GROUP BY
303      C.CustomerID, C.Name
304  ORDER BY
305      TotalSpending DESC
306  LIMIT 5;
307
308 |
```

Result Grid | Filter Rows:  Export: Wrap Cell Content: Fetch rows:

	CustomerID	Name	TotalSpending
▶	2	Jane Smith	1664.96
	1	John Doe	1634.96
	4	Alice Brown	1119.96
	5	Charlie Davis	1064.97
	3	Bob Johnson	634.97

```
309      -- 2. Monthly sales trend for the past 6 months
310
311 • SELECT
312      DATE_FORMAT(OrderDate, '%Y-%m') AS Month,
313      SUM(TotalAmount) AS MonthlySales
314  FROM
315      Orders
316  WHERE
317      OrderDate >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
318  GROUP BY
319      Month
320  ORDER BY
321      Month;
```

---

Result Grid			Filter Rows:	<input type="text"/>	Export:		Wrap Cell Content:	
	Month	MonthlySales						

```

324      -- 3. Best-selling product category in each month
325
326  ● WITH CategorySales AS (
327      SELECT
328          DATE_FORMAT(O.OrderDate, '%Y-%m') AS Month,
329          P.Category,
330          SUM(OD.Quantity * P.Price) AS TotalSales
331      FROM
332          Orders O
333      JOIN
334          OrderDetails OD ON O.OrderID = OD.OrderID
335      JOIN
336          Products P ON OD.ProductID = P.ProductID
337      GROUP BY
338          Month, P.Category
339      )
340      SELECT
341          cs.Month,
342          cs.Category,
343          cs.TotalSales
344      FROM
345          CategorySales cs
346      JOIN (
347          SELECT
348              Month,
349              MAX(TotalSales) AS MaxSales
350          FROM
351              CategorySales
352          GROUP BY
353              Month
354      ) AS maxSales ON cs.Month = maxSales.Month AND cs.TotalSales = maxSales.MaxSales
355      ORDER BY
356          cs.Month;

```

Result Grid | Filter Rows:

	Month	Category	TotalSales
▶	2023-01	Electronics	999.99
	2023-02	Electronics	599.99
	2023-03	Clothing	19.99
	2023-04	Electronics	999.99
	2023-05	Clothing	19.99
	2023-06	Electronics	599.99
	2023-07	Electronics	999.99
	2023-08	Electronics	599.99
	2023-09	Clothing	69.98
	2023-10	Electronics	999.99

```

359      -- 4. Customers who haven't made a purchase in the last 3 months
360
361 •  SELECT
362      C.CustomerID,
363      C.Name,
364      C.Email
365  FROM
366      Customers C
367  LEFT JOIN
368      Orders O ON C.CustomerID = O.CustomerID
369  WHERE
370      O.OrderDate IS NULL OR O.OrderDate < DATE_SUB(CURDATE(), INTERVAL 3 MONTH)
371  GROUP BY
372      C.CustomerID, C.Name, C.Email;
373
374
375
---
```

Result Grid | Filter Rows:  Export: Wrap Cell Content:

	CustomerID	Name	Email
▶	1	John Doe	john@example.com
	2	Jane Smith	jane@example.com
	3	Bob Johnson	bob@example.com
	4	Alice Brown	alice@example.com
	5	Charlie Davis	charlie@example.com

```

375      -- 5. Average order value and number of orders for each customer segment
376
377 •   SELECT
378     C.Segment,
379     COUNT(O.OrderID) AS NumberOfOrders,
380     AVG(O.TotalAmount) AS AverageOrderValue
381   FROM
382     Customers C
383   JOIN
384     Orders O ON C.CustomerID = O.CustomerID
385   GROUP BY
386     C.Segment;
387
---
```

	Segment	NumberOfOrders	AverageOrderValue
▶	Regular	6	555.816667
	Premium	4	696.230000

### Flowchart:

[Input Tables]

```

|
|-- Customers
|-- Products
|-- Orders
|-- OrderDetails
|
v
```

[CTEs]

```

|
|-- CustomerSpending
| |-- JOIN: Customers & Orders
| |-- GROUP BY: CustomerID, Name
| |-- Calculate: TotalSpending
|
|-- MonthlySales
| |-- Filter: Last 6 months
```

```
| |-- GROUP BY: Month
| |-- Calculate: MonthlySales
|
|-- BestSellingCategory
| |-- JOIN: Orders, OrderDetails & Products
| |-- GROUP BY: Month, Category
| |-- Calculate: TotalQuantity
| |-- Rank categories within each month
|
|-- InactiveCustomers
| |-- LEFT JOIN: Customers & Orders
| |-- GROUP BY: CustomerID, Name, Email
| |-- Filter: No orders in last 3 months
|
|-- CustomerSegmentAnalysis
| |-- LEFT JOIN: Customers & Orders
| |-- GROUP BY: Segment
| |-- Calculate: AvgOrderValue, NumberOfOrders
```

v

[Main Query]

```
|  
|-- Combine results from all CTEs using UNION ALL  
|-- Order results by Insight and Value  

|  
v
```

[Output]

Insight, Detail, Value

---

## Problem Statement:

You are working with a retail company that wants to analyze its sales data across different departments. They have provided you with a table containing employee sales information. Your task is to write a SQL query that accomplishes the following:

Calculate the total sales, average sales, and number of employees for each department.

Rank the departments based on their total sales.

Display this information in a single result set, ordered by total sales descending.

```
392 • CREATE TABLE employee_sales (
393     employee_id INT PRIMARY KEY,
394     employee_name VARCHAR(50),
395     department VARCHAR(50),
396     sales_amount DECIMAL(10, 2),
397     sales_date DATE
398 );
399
400 -- Sample data (insert statements) would go here
401
402 • INSERT INTO employee_sales (employee_id, employee_name, department, sales_amount, sales_date) VALUES
403     (1, 'John Doe', 'Electronics', 1500.00, '2023-01-15'),
404     (2, 'Jane Smith', 'Clothing', 2000.00, '2023-01-16'),
405     (3, 'Mike Johnson', 'Electronics', 1800.00, '2023-01-17'),
406     (4, 'Emily Brown', 'Home Goods', 1200.00, '2023-01-18'),
407     (5, 'David Lee', 'Clothing', 2200.00, '2023-01-19'),
408     (6, 'Sarah Wilson', 'Electronics', 1600.00, '2023-01-20'),
409     (7, 'Tom Harris', 'Home Goods', 1300.00, '2023-01-21'),
410     (8, 'Lisa Chen', 'Clothing', 1900.00, '2023-01-22');
411
```

```

413 • Ⓜ WITH department_sales AS (
414     SELECT
415         department,
416         SUM(sales_amount) AS total_sales,
417         AVG(sales_amount) AS average_sales,
418         COUNT(employee_id) AS number_of_employees,
419         AVG(sales_amount) / COUNT(DISTINCT sales_date) AS average_daily_sales
420     FROM
421         employee_sales
422     GROUP BY
423         department
424 )
425     SELECT
426         department,
427         total_sales,
428         average_sales,
429         number_of_employees,
430         average_daily_sales,
431         RANK() OVER (ORDER BY total_sales DESC) AS department_rank
432     FROM
433         department_sales
434     ORDER BY
435         total_sales DESC;
436

```

Result Grid						
	department	total_sales	average_sales	number_of_employees	average_daily_sales	department_rank
▶	Clothing	6100.00	2033.333333	3	677.777777777777	1
	Electronics	4900.00	1633.333333	3	544.4444444443	2
	Home Goods	2500.00	1250.000000	2	625.0000000000	3

# Day 25 - Spring Boot Application and Project Discussion

## Code Annotations related to a JPA entity in Java:

Primary Key (`@Id`) and Generated Value:

- The `@Id` annotation marks the primary key.
- The `@GeneratedValue(strategy = GenerationType.IDENTITY)` annotation indicates that the ID will be auto-generated using the database's identity column strategy.
- The field is of type `long` and named `id`.

One-to-Many Relationship:

- `@OneToMany(mappedBy = "restaurant")`: This annotation specifies a one-to-many relationship, likely between a `Restaurant` entity and another entity, where the relationship is managed by the "restaurant" field in the other entity.
- The cascade type is set to `CascadeType.ALL`, meaning that operations like persist, merge, remove, etc., will cascade to the related entities.

Column Mapping:

- The `@Column(name = "name")` annotation likely maps the entity's name field to a column in the database named "name".

Relationship Annotations:

- `@OneToOne`: Defines a one-to-one relationship between two entities.
- `@OneToMany`: Defines a one-to-many relationship, where one entity can be associated with multiple entities.
- `@ManyToOne`: Defines a many-to-one relationship, where many entities are related to one entity.
- `@ManyToMany`: Defines a many-to-many relationship, where many entities are related to many other entities.

Entity Class Example ([Post](#)):

- `@Entity`: Marks the class as a JPA entity.

- The `Post` class has an `id` field annotated with `@Id` and `@GeneratedValue` to auto-generate the primary key.

Many-to-One Relationship in `Post`:

- `@ManyToOne`: Indicates a many-to-one relationship within the `Post` entity.
- `@JoinColumn(name = "user_id")`: Specifies the foreign key column for the `ManyToOne` relationship, mapping it to the column `user_id` in the associated table.

Service Class:

- The `@Service` annotation marks the `UserService` class as a Spring service component.

Transactional Method:

- The method `addMoney(amount)` in the `UserService` class is annotated with `@Transactional`. This means the method will be executed within a database transaction.
- The flow of a transactional operation is illustrated:
  - Begin: The transaction begins when the method is called.
  - Commit: If the method completes successfully, the transaction is committed.
  - Rollback: If an exception is thrown during the method's execution, the transaction is rolled back, meaning any database changes made during the transaction are undone.

**Architecture of Spring boot Application:**

Client:

- The client sends an HTTP request to the Spring Boot application's controller layer.

Controller:

- This is the layer that handles incoming HTTP requests. It delegates the calls to appropriate services (such as `RestaurantService` or `ReviewService`) to perform the necessary business operations.

Spring Boot Application:

- The application is responsible for configuration and handling different operations related to restaurant and review management.

Services:

- RestaurantService: Handles business logic related to restaurants, including creating, reading, updating, and deleting (CRUD) operations and potentially handling asynchronous operations like queues.
- ReviewService: Handles CRUD operations related to reviews of restaurants.

Repositories:

- RestaurantRepository: Uses JPA to interact with the database for restaurant-related data operations.
- ReviewRepository: Uses JPA for review-related data persistence.

Database (MySQL):

- The MySQL database stores the data for both restaurant and review entities. JPA is used as the abstraction layer for database operations.

Layering (N-tier Architecture):

- The architecture is based on the N-tier design pattern:
  - Controller (Presentation Layer): Handles incoming HTTP requests and returns responses.
  - Business Logic Layer (Service Layer): Contains the core business logic for handling restaurant and review operations.
  - Data Access Layer (Repository): Handles interactions with the MySQL database through JPA.

## Restaurant Spring Boot Application:

```
demo > src > main > java > com > example > Restaurant.java > ...
1  package com.example;
2
3  import jakarta.persistence.*;
4  import java.util.List;
5
6  @Entity
7  @Table(name = "restaurant")
8  public class Restaurant {
9      @Id
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     private Long id;
12
13     @Column(nullable = false)
14     private String name;
15
16     @Column(nullable = false)
17     private String cusine;
18
19     @OneToMany(mappedBy = "restaurant", cascade = CascadeType.ALL, orphanRemoval = true)
20     private List<Review> reviews;
21
22     // Default constructor
23     public Restaurant() {}
24
25     // Parameterized constructor
26     public Restaurant(String name, String cusine) {
27         this.name = name;
28         this.cusine = cusine;
29     }
30
31     // Getters and Setters
32     public Long getId() {
33         return id;
34     }
35
36     public void setId(Long id) {
37         this.id = id;
38     }
39
40     public String getName() {
41         return name;
42     }
43
44     public void setName(String name) {
45         this.name = name;
46     }
47
48     public String getCusine() {
49         return cusine;
50     }
51
52     public void setCusine(String cusine) {
53         this.cusine = cusine;
54     }
55
56     public List<Review> getReviews() {
57         return reviews;
58     }
59
60     public void setReviews(List<Review> reviews) {
61         this.reviews = reviews;
62     }
63 }
64 }
```

```

demo > src > main > java > com > example > RestaurantController.java > updateRestaurant(Long, Restaurant)
  1 package com.example;
  2
  3 import org.springframework.beans.factory.annotation.Autowired;
  4 import org.springframework.http.ResponseEntity;
  5 import org.springframework.web.bind.annotation.*;
  6
  7 import java.util.List;
  8
  9 @RestController
10 @RequestMapping("/restaurants")
11 public class RestaurantController {
12
13     @Autowired
14     private RestaurantRepository restaurantRepository;
15
16     // Create a new restaurant
17     @PostMapping
18     public Restaurant createRestaurant(@RequestBody Restaurant restaurant) {
19         return restaurantRepository.save(restaurant);
20     }
21
22     // Get all restaurants
23     @GetMapping
24     public List<Restaurant> getAllRestaurants() {
25         return restaurantRepository.findAll();
26     }
27
28     // Get a single restaurant by ID
29     @GetMapping("/{id}")
30     public ResponseEntity<Restaurant> getRestaurantById(@PathVariable Long id) {
31         Restaurant restaurant = restaurantRepository.findById(id)
32             .orElseThrow(() -> new RuntimeException("Restaurant not found with id: " + id));
33         return ResponseEntity.ok(restaurant);
34     }
35
36     // Update a restaurant by ID
37     @PutMapping("/{id}")
38     public ResponseEntity<Restaurant> updateRestaurant(@PathVariable Long id, @RequestBody Restaurant restaurantDetails) {
39         Restaurant restaurant = restaurantRepository.findById(id)
40             .orElseThrow(() -> new RuntimeException("Restaurant not found with id: " + id));
41
42         restaurant.setName(restaurantDetails.getName());
43         restaurant.setCuisine(restaurantDetails.getCuisine());
44
45         Restaurant updatedRestaurant = restaurantRepository.save(restaurant);
46         return ResponseEntity.ok(updatedRestaurant);
47     }
48
49     // Delete a restaurant by ID
50     @DeleteMapping("/{id}")
51     public ResponseEntity<Void> deleteRestaurant(@PathVariable Long id) {
52         Restaurant restaurant = restaurantRepository.findById(id)
53             .orElseThrow(() -> new RuntimeException("Restaurant not found with id: " + id));
54
55         restaurantRepository.delete(restaurant);
56         return ResponseEntity.noContent().build();
57     }
58
59

```

```
demo > src > main > java > com > example > RestaurantRepository.java > ...
1 package com.example;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 | Ctrl+L to chat, Ctrl+K to generate
5 public interface RestaurantRepository extends JpaRepository<Restaurant, Long> {
6
7 }
8
```

```
demo > src > main > java > com > example > RestaurantService.java > RestaurantService
1 package com.example;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 @Service
10 public class RestaurantService {
11
12     @Autowired
13     private RestaurantRepository restaurantRepository;
14
15     public List<Restaurant> getAllRestaurants() {
16         return restaurantRepository.findAll();
17     }
18
19     public Optional<Restaurant> getRestaurantById(Long id) {
20         return restaurantRepository.findById(id);
21     }
22
23     public Restaurant saveRestaurant(Restaurant restaurant) {
24         return restaurantRepository.save(restaurant);
25     }
26
27     public void deleteRestaurant(Long id) {
28         restaurantRepository.deleteById(id);
29     }
30 }
31
```

```

demo > src > main > java > com > example > Review.java > ...
1 package com.example;
2
3 import jakarta.persistence.*;
4
5 @Entity
6 @Table(name = "review")
7 public class Review {
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    private Long id;
11
12    @Column(nullable = false)
13    private String reviewerName;
14
15    @Column(nullable = false)
16    private int rating;
17
18    @ManyToOne
19    @JoinColumn(name = "restaurant_id", nullable = false)
20    private Restaurant restaurant;
21
22    @Column(nullable = false)
23    private String comment;
24
25    // Default constructor
26    public Review() {}
27
28    // Parameterized constructor
29    public Review(String reviewerName, int rating, Restaurant restaurant, String comment) {
30        this.reviewerName = reviewerName;
31        this.rating = rating;
32        this.restaurant = restaurant;
33        this.comment = comment;
34    }
35
36    // Getters and Setters
37    public Long getId() {
38        return id;
39    }
40
41    public void setId(Long id) {
42        this.id = id;
43    }
44
45    public String getReviewerName() {
46        return reviewerName;
47    }
48
49    public void setReviewerName(String reviewerName) {
50        this.reviewerName = reviewerName;
51    }
52
53    public int getRating() {
54        return rating;
55    }
56
57    public void setRating(int rating) {
58        this.rating = rating;
59    }
60
61    public Restaurant getRestaurant() {
62        return restaurant;
63    }
64
65    public void setRestaurant(Restaurant restaurant) {
66        this.restaurant = restaurant;
67    }
68
69    public String getComment() {
70        return comment;
71    }
72
73    public void setComment(String comment) {
74        this.comment = comment;
75    }
76}
77
```

Ctrl+Shift+C Ctrl+Shift+G generate

```

demo > src > main > java > com > example > ReviewController.java > ReviewController > reviewRepository
 1 package com.example;
 2
 3 import org.springframework.beans.factory.annotation.Autowired;
 4 import org.springframework.http.ResponseEntity;
 5 import org.springframework.web.bind.annotation.*;
 6
 7 import java.util.List;
 8
 9 @RestController
10 @RequestMapping("/reviews")
11 public class ReviewController {
12
13     @Autowired
14     private ReviewRepository reviewRepository;
15
16     @Autowired
17     private RestaurantRepository restaurantRepository;
18
19     // Create a new review for a restaurant
20     @PostMapping("/{restaurantId}")
21     public Review createReview(@PathVariable Long restaurantId, @RequestBody Review review) {
22         Restaurant restaurant = restaurantRepository.findById(restaurantId)
23             .orElseThrow(() -> new RuntimeException("Restaurant not found with id: " + restaurantId));
24         review.setRestaurant(restaurant);
25         return reviewRepository.save(review);
26     }
27
28     // Get all reviews for a specific restaurant
29     @GetMapping("/restaurant/{restaurantId}")
30     public List<Review> getReviewsByRestaurant(@PathVariable Long restaurantId) {
31         Restaurant restaurant = restaurantRepository.findById(restaurantId)
32             .orElseThrow(() -> new RuntimeException("Restaurant not found with id: " + restaurantId));
33         return restaurant.getReviews();
34     }
35
36     // Get a single review by ID
37     @GetMapping("/{id}")
38     public ResponseEntity<Review> getReviewById(@PathVariable Long id) {
39         Review review = reviewRepository.findById(id)
40             .orElseThrow(() -> new RuntimeException("Review not found with id: " + id));
41         return ResponseEntity.ok(review);
42     }
43
44     // Delete a review by ID
45     @DeleteMapping("/{id}")
46     public ResponseEntity<Void> deleteReview(@PathVariable Long id) {
47         Review review = reviewRepository.findById(id)
48             .orElseThrow(() -> new RuntimeException("Review not found with id: " + id));
49
50         reviewRepository.delete(review);
51         return ResponseEntity.noContent().build();
52     }
53 }
54

```

```
demo > src > main > java > com > example > ReviewRepository.java > ...
1 package com.example;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface ReviewRepository extends JpaRepository<Review, Long> {
6
7     Ctrl+L to chat, Ctrl+K to generate
8 }
```

```
demo > src > main > java > com > example > ReviewService.java > ...
1 package com.example;
2
3 import java.util.List;
4 import java.util.Optional;
5 import org.springframework.stereotype.Service;
6 import com.example.Review;
7
8 @Service
9 public interface ReviewService {
10     List<Review> getAllReviews();
11     Optional<Review> getReviewById(Long id);
12     Review saveReview(Review review);
13     void deleteReview(Long id);
14 }
```

HTTP <http://localhost:8081/api/restaurants>

POST



http://localhost:8081/api/restaurants

Params

Authorization

Headers (9)

Body •

Scripts

Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON**

```
1 {
2     "name": "Pasta Palace",
3     "cuisine": "Italian"
4 }
5
```

## Day 26 - Spring Boot Concepts

```
demo>src>main>java>com>example> WeatherController.java > WeatherController.java
 1 package com.example;
 2
 3 import org.springframework.web.bind.annotation.RestController;
 4
 5 import java.util.Map;
 6 import org.springframework.web.bind.annotation.GetMapping;
 7 import org.springframework.web.bind.annotation.PathVariable;
 8 import org.springframework.web.bind.annotation.RequestMapping;
 9 import org.springframework.web.bind.annotation.RequestParam;
10 import org.springframework.web.bind.annotation.PostMapping;
11 import org.springframework.web.bind.annotation.DeleteMapping;
12
13 import javax.validation.constraints.NotBlank;
14 import org.springframework.beans.factory.annotation.Autowired;
15 import org.springframework.validation.annotation.Validated;
16 import java.util.List;
17 import org.springframework.http.ResponseEntity;
18 import org.springframework.web.bind.annotation.PutMapping;
19 import org.springframework.web.bind.annotation.RequestBody;
20 import org.springframework.http.HttpStatus;
21 import javax.validation.Valid;
22
23 @RestController
24 @RequestMapping("/api/weather")
25 @Validated
26 public class WeatherController {
27
28     private final WeatherService weatherService;
29     @Autowired
30     public WeatherController(WeatherService weatherService){
31         this.weatherService = weatherService;
32     }
33
34     @GetMapping("/all")
35     public List<WeatherRecord> getAllWeather(){
36         return weatherService.getAllWeather();
37     }
38
39     @GetMapping("/{city}")
40     public ResponseEntity<WeatherRecord> getWeatherByCity(@PathVariable @NotBlank String city) {
41         WeatherRecord weather = weatherService.getWeatherByCity(city);
42         return weather != null ? ResponseEntity.ok(weather) : ResponseEntity.notFound().build();
43     }
44
45     @PostMapping("/create")
46     public WeatherRecord addWeather(@Valid @RequestBody WeatherRequestDTO weatherRequestDTO){
47         return weatherService.addWeather(weatherRequestDTO);
48     }
49
50     /*
51     @PutMapping("/{id}")
52     public ResponseEntity<WeatherRecord> updateWeather(@PathVariable Long id, @RequestBody WeatherRecord weatherRecord){
53         if(!weatherService.getAllWeather().stream().anyMatch(weather -> weather.getId().equals(id))){
54             return ResponseEntity.notFound().build();
55         }
56         weatherRecord.setId(id);
57         return ResponseEntity.ok(weatherService.updateWeather(weatherRecord));
58     }
59     @DeleteMapping("/{id}")
60     public ResponseEntity<Void> deleteWeather(@PathVariable Long id){
61         if(!weatherService.getAllWeather().stream().anyMatch(weather -> weather.getId().equals(id))){
62             return ResponseEntity.notFound().build();
63         }
64         weatherService.deleteWeather(id);
65         return ResponseEntity.noContent().build();
66     }
67 }
```

```
demo > src > main > java > com > example > J WeatherException.java
 1 package com.example;
 2
 3 import java.util.*;
 4
 5 public class WeatherException extends RuntimeException{
 6     public WeatherException(String message){
 7         super(message);
 8     }
 9     public WeatherException(String message, Throwable cause){
10         super(message, cause);
11     }
12 }
```

```
demo > src > main > java > com > example > J WeatherRecord.java > () com.example
 1 package com.example;
 2
 3 import jakarta.persistence.*;
 4
 5 @Entity
 6 @Table(name="weather_records")
 7 public class WeatherRecord {
 8     @Id
 9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10     private Long id;
11     @Column(nullable = false,unique = true)
12     private String city;
13     @Column(nullable = false)
14     private int temperature;
15     @Column(nullable = false)
16     private String weatherCondition;
17
18     public WeatherRecord(){
19     }
20
21     public void initialize(String city, int temperature){
22         this.city = city;
23         this.temperature = temperature;
24     }
25
26     public Long getId(){
27         return id;
28     }
29     public void setId(Long id){
30         this.id = id;
31     }
32
33     public String getCity(){
34         return city;
35     }
36
37     public String getWeatherCondition(){
38         return weatherCondition;
39     }
40     public void setCity(String city){
41         this.city = city;
42     }
43     public void setWeatherCondition(String weatherCondition){
44         this.weatherCondition = weatherCondition;
45     }
46     public void setTemperature(int temperature){
47         this.temperature = temperature;
48     }
49     public int getTemperature(){
50         return temperature;
51     }
52 }
53 }
```

```
demo > src > main > java > com > example > WeatherRequestDTO.java
 1 package com.example;
 2
 3 import jakarta.validation.constraints.*;
 4 public class WeatherRequestDTO {
 5     @NotBlank(message = "City is required and cannot be empty")
 6     @Size(min = 3, max = 20, message = "City must be between 3 and 20 characters")
 7     private String city;
 8
 9     @Min(value = -50, message = "Temperature cannot be less than -50")
10     @Max(value = 50, message = "Temperature cannot be greater than 50")
11     private int temperature;
12
13     @NotBlank(message = "Weather condition is required and cannot be empty")
14     @Size(min = 3, max = 20, message = "Weather condition must be between 3 and 20 characters")
15     @Pattern(regexp = "^\\s*(sunny|cloudy|rainy|snowy)\\s*$", message = "Weather condition must be sunny, cloudy, rainy, or snowy")
16     private String weatherCondition;
17
18     //getters and setters
19     public String getCity(){
20         return city;
21     }
22     public void setCity(String city){
23         this.city = city;
24     }
25     public int getTemperature(){
26         return temperature;
27     }
28     public void setTemperature(int temperature){
29         this.temperature = temperature;
30     }
31     public String getWeatherCondition(){
32         return weatherCondition;
33     }
34     public void setWeatherCondition(String weatherCondition){
35         this.weatherCondition = weatherCondition;
36     }
37 }
38 }
```

```
demo > src > main > java > com > example > J WeatherService.java > WeatherService
  1 package com.example;
  2
  3 import java.util.HashMap;
  4 import java.util.Map;
  5 import org.springframework.beans.factory.annotation.Autowired;
  6 import org.springframework.stereotype.Service;
  7 import java.util.List;
  8 import org.springframework.validation.annotation.Validated;
  9 import org.springframework.transaction.annotation.Transactional;
 10 import com.example.WeatherRequestDTO;
 11 @Service
 12 @Validated
 13 public class WeatherService {
 14     private final WeatherRepository weatherRepository;
 15
 16     private final WeatherValidator weatherValidator;
 17     @Autowired
 18     public WeatherService(WeatherRepository weatherRepository, WeatherValidator weatherValidator){
 19         this.weatherRepository = weatherRepository;
 20         this.weatherValidator = weatherValidator;
 21     }
 22
 23     public List<WeatherRecord> getAllWeather(){
 24         return weatherRepository.findAll();
 25     }
 26
 27     public WeatherRecord getWeatherByCity(String city){
 28         return weatherRepository.findByCity(city).orElseThrow(() -> new RuntimeException(message:"City not found"));
 29     }
 30
 31     private WeatherRecord convertToEntity(WeatherRequestDTO weatherRecordDTO){
 32         WeatherRecord weatherRecord = new WeatherRecord();
 33         weatherRecord.setCity(weatherRecordDTO.getCity());
 34         weatherRecord.setTemperature(weatherRecordDTO.getTemperature());
 35         weatherRecord.setWeatherCondition(weatherRecordDTO.getWeatherCondition());
 36         return weatherRecord;
 37     }
 38     @Transactional
 39     public WeatherRecord addWeather(WeatherRequestDTO weatherRecordDTO){
 40         weatherValidator.validateWeatherCondition(weatherRecordDTO);
 41
 42         WeatherRecord weatherRecord = convertToEntity(weatherRecordDTO);
 43         return weatherRepository.save(weatherRecord);
 44     }
 45
 46     public void deleteWeather(Long id){
 47         weatherRepository.deleteById(id);
 48     }
 49     public WeatherRecord updateWeather(WeatherRecord weatherRecord){
 50         return weatherRepository.save(weatherRecord);
 51     }
 52 }
```

```
demo > src > main > java > com > example > WeatherValidationConstants.java > WeatherValidationConstants
 1 package com.example;
 2
 3 public class WeatherValidationConstants {
 4     public static final int MIN_TEMPERATURE = -50;
 5     public static final int MAX_TEMPERATURE = 50;
 6
 7     public static final String WEATHER_CONDITION_PATTERN = "^(sunny|cloudy|rainy|snowy)$";
 8     public static final int CITY_MIN_LENGTH = 3;
 9     public static final int CITY_MAX_LENGTH = 20;
10
11     public static final String CITY_BLANK_MESSAGE = "City is required and cannot be empty";
12     public static final String CITY_SIZE_MESSAGE = "City must be between 3 and 20 characters";
13     public static final String CITY_PATTERN_MESSAGE = "City must be sunny, cloudy, rainy, or snowy";
14
15     public static final String TEMPERATURE_MIN_MESSAGE = "Temperature cannot be less than -50";
16     public static final String TEMPERATURE_MAX_MESSAGE = "Temperature cannot be greater than 50";
17
18     public static final String WEATHER_CONDITION_BLANK_MESSAGE = "Weather condition is required and cannot be empty";
19     public static final String WEATHER_CONDITION_SIZE_MESSAGE = "Weather condition must be between 3 and 20 characters";
20     public static final String WEATHER_CONDITION_PATTERN_MESSAGE = "Weather condition must be sunny, cloudy, rainy, or snowy";
21 }
```

```
demo > src > main > java > com > example > WeatherValidationException.java > WeatherValidationException
 1 package com.example;
 2 import java.util.*;
 3
 4 public class WeatherValidationException extends WeatherException{
 5     private final List<String> violations;
 6
 7     public WeatherValidationException(String message, List<String> violations){
 8         super(message);
 9         this.violations = violations.isEmpty()
10             ? List.of()
11             : List.of(String.join(delimiter:" ", violations));
12     }
13     public WeatherValidationException(List<String> violations){
14         super("Weather validation failed: " + String.join(delimiter:" ", violations));
15         this.violations = new ArrayList<>(violations);
16     }
17
18     public List<String> getViolations(){
19         return violations;
20     }
21
22
23
24 }
```

```
demo > src > main > java > com > example > WeatherValidator.java > WeatherValidator
 1 package com.example;
 2
 3 import org.springframework.stereotype.Component;
 4 import java.util.List;
 5 import java.util.ArrayList;
 6
 7 @Component
 8 public class WeatherValidator {
 9     public void validateWeatherCondition(WeatherRequestDTO weatherRequestDTO){
10         List<String> violations = new ArrayList<>();
11
12         if(weatherRequestDTO.getTemperature() > 30 && "Snowy".equals(weatherRequestDTO.getWeatherCondition())){
13             violations.add("Snowy weather is not allowed when temperature is above 30 degrees");
14         }
15
16         if(weatherRequestDTO.getTemperature() < 0 && "Sunny".equals(weatherRequestDTO.getWeatherCondition())){
17             violations.add("Sunny weather is not allowed when temperature is below 0 degrees");
18         }
19
20         if(violations.isEmpty()){
21             return;
22         }
23         throw new WeatherValidationException(String.join(delimiter:" ", violations), violations);
24     }
25 }
```

## **Spring Bean Scopes**

### **## 1. Singleton Scope (Default)**

- Only ONE instance is created for the entire application
- Same instance is shared across all requests
- Default scope in Spring
- Good for: Stateless beans, Services, Repositories

Example:

```
java
@Service
// or @Service(@Scope("singleton")) - not needed as it's default
public class CounterService {
    private int counter = 0; // This value is shared

    public void increment() {
        counter++;
    }

    public int getCount() {
        return counter;
    }
}
```

Usage Pattern:

- All users see the same counter value
- Counter increments globally

### **## 2. Prototype Scope**

- New instance created every time requested
- Good for: Stateful beans, Heavy resources
- Each injection gets its own instance

Example:

```
java
@Component
@Scope("prototype")
public class UserRequest {
    private final String requestId = UUID.randomUUID().toString();
    private final LocalDateTime created = LocalDateTime.now();

    public String getRequestId() {
        return requestId;
    }
}
```

#### Usage Pattern:

- Each time getBean() is called, new instance
- Each instance has its own state

#### ## 3. Request Scope

- New instance for each HTTP request
- Instance lives only for the duration of HTTP request
- Good for: Request-specific data, Logging

#### Example:

```
java
@Component
@RequestScope // same as @Scope(value = WebApplicationContext.SCOPE_REQUEST)
public class RequestLogger {
    private final String requestId = UUID.randomUUID().toString();
    private List<String> logs = new ArrayList<>();

    public void addLog(String message) {
        logs.add(message);
    }
}
```

Usage Pattern:

- New instance per HTTP request
- Data dies after request completes

#### ## 4. Session Scope

- New instance for each user session
- Instance lives for entire user session
- Good for: Shopping carts, User preferences

Example:

```
java
@Component
@SessionScope
public class ShoppingCart {
    private List<String> items = new ArrayList<>();

    public void addItem(String item) {
        items.add(item);
    }

    public List<String> getItems() {
        return items;
    }
}
```

Usage Pattern:

- Persists across requests for same user
- Different users get different instances

## Day 27 - Spring Continues...

### **Spring Security and JWT (JSON Web Token) authentication flow:**

Client: The process starts with a login request sent from the client.

AuthController: This controller handles the login request. It validates the credentials.

UserService: The `AuthController` uses a `UserService` to interact with the database (`DB`) to check the user's credentials.

JWT Token Generation: If the credentials are valid, the `UserService` generates a JWT token.

Token Return: The JWT token is returned to the client.

Client Sends Token: After receiving the JWT, the client includes the token in subsequent requests.

JWT Filter: This filter intercepts the request and validates the JWT token.

- If the token is valid, the request is passed forward.
- If invalid, it is rejected.

Token Validation: Once validated, the user's roles and permissions are extracted from the JWT, allowing access to protected resources.

### **The JWT authentication flow includes additional details related to Spring Security context and protected resources:**

1. **Client:** The process begins with the client attempting to log in.
2. **Login & Authentication Controller:**
  - The client sends a login request to the `AuthController`.
  - The `AuthController` verifies the credentials (`Verify Cr`) using the `UserRepo` (User Repository).
3. **JWT Token Creation:**
  - If the credentials are correct, a JWT token is generated and returned to the client.
  - The token contains the user's role, such as a general role (e.g., admin or regular user).

---

**4. Client Sends Token:**

- For subsequent requests, the client includes the JWT in the headers to access protected resources.

**5. JWT Filter:**

- The JWT Filter intercepts the request and verifies the token. If valid, it proceeds with setting the user's security context.

**6. Security Context:**

- The JWT token is used to set the authentication in the `SecurityContext`. This is where the user's details and roles are stored.

**7. Access Protected Resource:**

- Once the security context is set, the user is allowed access to the protected resource, assuming the user has the proper role or permissions.

```

demo > src > main > java > com > example > JwtUtil.java > {} com.example
  1 // JwtUtil.java
  2 package com.example;
  3
  4 import io.jsonwebtoken.Claims;
  5 import io.jsonwebtoken.Jwts;
  6 import io.jsonwebtoken.SignatureAlgorithm;
  7 import org.springframework.stereotype.Component;
  8
  9 import java.util.Date;
 10 import java.util.HashMap;
 11 import java.util.Map;
 12
 13 @Component
 14 public class JwtUtil {
 15     private String SECRET_KEY = "my_secret_key";
 16
 17     public String generateToken(String username) {
 18         Map<String, Object> claims = new HashMap<>();
 19         return createToken(claims, username);
 20     }
 21
 22     private String createToken(Map<String, Object> claims, String subject) {
 23         return Jwts.builder()
 24             .setClaims(claims)
 25             .setSubject(subject)
 26             .setIssuedAt(new Date(System.currentTimeMillis()))
 27             .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 10)) // 10 hours expiration
 28             .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
 29             .compact();
 30     }
 31
 32     public Boolean validateToken(String token, String username) {
 33         final String extractedUsername = extractUsername(token);
 34         return (extractedUsername.equals(username) && !isTokenExpired(token));
 35     }
 36
 37     public String extractUsername(String token) {
 38         return extractAllClaims(token).getSubject();
 39     }
 40
 41     private Claims extractAllClaims(String token) {
 42         return Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token).getBody();
 43     }
 44
 45     private Boolean isTokenExpired(String token) {
 46         return extractAllClaims(token).getExpiration().before(new Date());
 47     }
 48 }
 49

```

```
demo > src > main > java > com > example > LoginRequest.java > ...
1 // LoginRequest.java
2 package com.example;
3
4 public class LoginRequest {
5     private String username;
6     private String password;
7
8     // Getters and Setters
9
10    Ctrl+L to chat, Ctrl+K to generate
```

```
demo > src > main > java > com > example > LoginResponse.java > LoginResponse > getRole()
1 // LoginResponse.java
2 package com.example.security.dto;
3
4 public class LoginResponse {
5     private String token;
6     private String username;
7     private String role;
8
9     public LoginResponse(String token, String username, String role) {
10         this.token = token;
11         this.username = username;
12         this.role = role;
13     }
14
15     // Getters and setters
16     public String getToken() {
17         return token;
18     }
19
20     public void setToken(String token) {
21         this.token = token;
22     }
23
24     public String getUsername() {
25         return username;
26     }
27
28     public void setUsername(String username) {
29         this.username = username;
30     }
31
32     public String getRole() {
33         return role;
34     }
35
36     public void setRole(String role) {
37         this.role = role;
38     }
39 }
```

```

demo > src > main > java > com > example > SecurityConfig.java > ...
1 package com.example;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.security.authentication.AuthenticationManager;
7 import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
8 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
9 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
10 import org.springframework.security.config.http.SessionCreationPolicy;
11 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
12 import org.springframework.security.crypto.password.PasswordEncoder;
13 import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
14
15 @Configuration
16 @EnableWebSecurity
17 public class SecurityConfig extends WebSecurityConfigurerAdapter {
18
19     @Autowired
20     private JwtFilter jwtFilter;
21
22     @Autowired
23     private UserService userService;
24
25     @Override
26     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
27         auth.userDetailsService(username -> userService.findByUsername(username));
28     }
29
30     @Override
31     protected void configure(HttpSecurity http) throws Exception {
32         http.csrf().disable()
33             .authorizeRequests().antMatchers("/login").permitAll()
34             .anyRequest().authenticated()
35             .and()
36             .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
37
38         http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);
39     }
40
41     @Bean
42     public PasswordEncoder passwordEncoder() {
43         return new BCryptPasswordEncoder();
44     }
45
46     @Bean
47     @Override
48     public AuthenticationManager authenticationManagerBean() throws Exception {
49         return super.authenticationManagerBean();
50     }
51 }
52 ⚡ Ctrl+L to chat, Ctrl+K to generate
53

```

```
demo > src > main > java > com > example > User.java > User > username
 1 package com.example;
 2
 3 import javax.persistence.Entity;
 4 import javax.persistence.Id;
 5 import javax.persistence.Table;
 6
 7 @Entity
 8 @Table(name = "users")
 9 public class User {
10     @Id
11     private String username;
12     private String password;
13     private String role;
14
15     // Getters and Setters
16 }
17
```

```
demo > src > main > java > com > example > UserRepository.java > {} com.example
● 1 // UserRepository.java
 2 package com.example;
 3
 4 import org.springframework.data.jpa.repository.JpaRepository;
 5
 6 public interface UserRepository extends JpaRepository<User, String> {
 7     User findByUsername(String username);
 8 }
 9
```

```
demo > src > main > java > com > example > UserService.java > ...
1 // UserService.java
2 package com.example;
3
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Service;
6
7 @Service
8 public class UserService {
9     @Autowired
10    private UserRepository userRepository;
11
12    public User findByUsername(String username) {
13        return userRepository.findByUsername(username);
14    }
15
16    public void saveUser(User user) {
17        userRepository.save(user);
18    }
19
20 }
```

Ctrl+L to chat, Ctrl+K to generate

```

demo > src > main > java > com > example > JwtFilter.java > ...
1 package com.example;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
5 import org.springframework.security.core.context.SecurityContextHolder;
6 import org.springframework.security.core.userdetails.UserDetails;
7 import org.springframework.security.core.userdetails.UserDetailsService;
8 import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
9 import org.springframework.stereotype.Component;
10 import org.springframework.web.filter.OncePerRequestFilter;
11
12 import javax.servlet.FilterChain;
13 import javax.servlet.http.HttpServletRequest;
14 import javax.servlet.http.HttpServletResponse;
15 import java.io.IOException;
16
17 @Component
18 public class JwtFilter extends OncePerRequestFilter {
19
20     @Autowired
21     private JwtUtil jwtUtil;
22
23     @Autowired
24     private UserService userService;
25
26     @Override
27     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
28             throws IOException, javax.servlet.ServletException {
29
30         String authorizationHeader = request.getHeader("Authorization");
31
32         String username = null;
33         String jwt = null;
34
35         if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
36             jwt = authorizationHeader.substring(7);
37             username = jwtUtil.extractUsername(jwt);
38         }
39
40         if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
41             User user = userService.findByUsername(username);
42
43             if (jwtUtil.validateToken(jwt, username)) {
44                 UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(
45                     user, null, new ArrayList<>());
46                 authenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
47                 SecurityContextHolder.getContext().setAuthentication(authenticationToken);
48             }
49         }
50         chain.doFilter(request, response);
51     }
52 }
53 
```

Ctrl+L to chat, Ctrl+K to generate

# Day 28 - Working with Nodejs and Angular

## Installation of Nodejs and Angular

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>

```
PS C:\WINDOWS\system32> Get -ExecutionPolicy
```

```
Get : The term 'Get' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
```

```
At line:1 char:1
```

```
+ Get -ExecutionPolicy
```

```
+ ~~~
```

```
+ CategoryInfo : ObjectNotFound: (Get:String) [], CommandNotFoundException
```

```
+ FullyQualifiedErrorId : CommandNotFoundException
```

```
PS C:\WINDOWS\system32> Get-ExecutionPolicy
```

```
Restricted Set-ExecutionPolicy AllSigned
```

```
>> C:\WINDOWS\system32>
```

Execution Policy Change

The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose you to the security risks described in the about\_Execution\_Policies help topic at <https://go.microsoft.com/fwlink/?LinkID=135170>. Do you want to change the execution policy?

[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): Y

```
PS C:\WINDOWS\system32> Get-ExecutionPolicy
```

```
>>
```

AllSigned

```
PS C:\WINDOWS\system32> Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

```
>>
```

WARNING: 'choco' was found at 'C:\ProgramData\chocolatey\bin\choco.exe'.

WARNING: An existing Chocolatey installation was detected. Installation will not continue. This script will not overwrite existing installations.

If there is no Chocolatey installation at 'C:\ProgramData\chocolatey', delete the folder and attempt the installation again.

Please use choco upgrade chocolatey to handle upgrades of Chocolatey itself.

If the existing installation is not functional or a prior installation did not complete, follow these steps:

- Backup the files at the path listed above so you can restore your previous installation if needed.
- Remove the existing installation manually.
- Rerun this installation script.
- Reinstall any packages previously installed, if needed (refer to the lib folder in the backup).

Once installation is completed, the backup folder is no longer needed and can be deleted.

```
PS C:\WINDOWS\system32> choco --version
```

```
>>
```

2.3.0

```
PS C:\WINDOWS\system32> choco install nodejs-lts --version="20.18.0"
```

Chocolatey v2.3.0

Installing the following packages:

nodejs-lts

By installing, you accept licenses for the packages.

```
Downloading package from source 'https://community.chocolatey.org/api/v2/'
```

Progress: Downloading nodejs-lts 20.18.0... 100%

nodejs-lts v20.18.0 [Approved]  
nodejs-lts package files install completed. Performing other installation steps.  
The package nodejs-lts wants to run 'chocolateyinstall.ps1'.  
Note: If you don't run this script, the installation will fail.  
Note: To confirm automatically next time, use '-y' or consider:  
choco feature enable -n allowGlobalConfirmation  
Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): A

Installing 64 bit version  
Installing nodejs-lts...  
nodejs-lts has been installed.  
nodejs-lts may be able to be automatically uninstalled.  
Environment Vars (like PATH) have changed. Close/reopen your shell to  
see the changes (or in powershell/cmd.exe just type 'refreshenv').  
The install of nodejs-lts was successful.  
Software installed as 'MSI', install location is likely default.

Chocolatey installed 1/1 packages.  
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).  
PS C:\WINDOWS\system32> choco -version  
Parsing -version resulted in error (converted to warning):  
Cannot bundle unregistered option '-e'.  
This is a listing of all of the different things you can pass to choco.

#### Options and Switches

-v, --version  
Version - Prints out the Chocolatey version.

#### Commands

- \* apikey - retrieves, saves or deletes an API key for a particular source
- \* cache - Manage the local HTTP caches used to store queries (v2.1.0+)
- \* config - Retrieve and configure config file settings
- \* export - exports list of currently installed packages
- \* feature - view and configure choco features
- \* features - view and configure choco features (alias for feature)
- \* find - searches remote packages (alias for search)
- \* help - displays top level help information for choco
- \* info - retrieves package information. Shorthand for choco search pkgname --exact --verbose
- \* install - installs packages using configured sources
- \* list - lists local packages
- \* new - creates template files for creating a new Chocolatey package
- \* outdated - retrieves information about packages that are outdated. Similar to upgrade all --noop
- \* pack - packages nuspec, scripts, and other Chocolatey package resources into a nupkg file
- \* pin - suppress upgrades for a package
- \* push - pushes a compiled nupkg to a source
- \* rule - view or list implemented package rules (v2.3.0+)
- \* search - searches remote packages
- \* setapikey - retrieves, saves or deletes an API key for a particular source (alias for apikey)
- \* source - view and configure default sources
- \* sources - view and configure default sources (alias for source)
- \* template - get information about installed templates
- \* templates - get information about installed templates (alias for template)
- \* uninstall - uninstalls a package
- \* unpackself - [DEPRECATED] will be removed in v3.0.0 - re-installs Chocolatey base files
- \* upgrade - upgrades packages from various sources

Please run chocolatey with `choco command -help` for specific help on each command.

## How To Pass Options / Switches

You can pass options and switches in the following ways:

- \* Unless stated otherwise, an option/switch should only be passed one time. Otherwise you may find weird/non-supported behavior.
- \* `-`, `/`, or `--` (one character switches should not use `--)
- \* **Option Bundling / Bundled Options**: One character switches can be bundled. e.g. `-d` (debug), `-f` (force), `-v` (verbose), and `-y` (confirm yes) can be bundled as `-dfvy`.
- \* NOTE: If `debug` or `verbose` are bundled with local options (not the global ones above), some logging may not show up until after the local options are parsed.
- \* **Use Equals**: You can also include or not include an equals sign `=` between options and values.
- \* **Quote Values**: When you need to quote an entire argument, such as when using spaces, please use a combination of double quotes and apostrophes ("value"). In cmd.exe you can just use double quotes ("value") but in powershell.exe you should use backticks (` ` "value" ` `) or apostrophes ('value'). Using the combination allows for both shells to work without issue, except for when the next section applies.
- \* **Pass quotes in arguments**: When you need to pass quoted values to something like a native installer, you are in for a world of fun. In cmd.exe you must pass it like this: `-ia "/yo=""Spaces spaces""". In PowerShell.exe, you must pass it like this: `-ia '/yo=""Spaces spaces""'. No other combination will work. In PowerShell.exe if you are on version v3+, you can try `--%` before `-ia` to just pass the args through as is, which means it should not require any special workarounds.
- \* **Periods in PowerShell**: If you need to pass a period as part of a value or a path, PowerShell doesn't always handle it well. Please quote those values using "Quote Values" section above.
- \* Options and switches apply to all items passed, so if you are installing multiple packages, and you use `--version=1.0.0`, choco is going to look for and try to install version 1.0.0 of every package passed. So please split out multiple package calls when wanting to pass specific options.

## Scripting / Integration - Best Practices / Style Guide

When writing scripts, such as PowerShell scripts passing options and switches, there are some best practices to follow to ensure that you don't run into issues later. This also applies to integrations that are calling Chocolatey and parsing output. Chocolatey **uses** PowerShell, but it is an exe, so it cannot return PowerShell objects.

Following these practices ensures both readability of your scripts AND compatibility across different versions and editions of Chocolatey. Following this guide will ensure your experience is not frustrating based on choco not receiving things you think you are passing to it.

- \* For consistency, always use `choco`, not `choco.exe`.
- \* Always have the command as the first argument to `choco`. e.g. `choco install`, where `install` is the command.
- \* If there is a subcommand, ensure that is the second argument. e.g. `choco source list`, where `source` is the command and `list` is the

subcommand.

- \* Typically the subject comes next. If installing packages, the subject would be the package names, e.g. `choco install pkg1 pkg2`.
- \* Never use 'nupkg' or point directly to a nupkg file UNLESS using 'choco push'. Use the source folder instead, e.g. `choco install <package id> --source="c:\folder\with\package"" instead of 'choco install DoNotDoThis.1.0.nupkg' or `choco install DoNotDoThis --source="c:\folder\with\package\DoNotDoThis.1.0.nupkg""`.
- \* Switches and parameters are called simply options. Options come after the subject. e.g. `choco install pkg1 --debug --verbose`.
- \* Never use the force option (`--force` / '-f') in scripts (or really otherwise as a default mode of use). Force is an override on Chocolatey behavior. If you are wondering why Chocolatey isn't doing something like the documentation says it should, it's likely because you are using force. Stop.
- \* Always use full option name. If the short option is '-n', and the full option is '--name', use '--name'. The only acceptable short option for use in scripts is '-y'. Find option names in help docs online or through `choco -?` / `choco [Command Name] -?`.
- \* For scripts that are running automated, always use '-y'. Do note that even with '-y' passed, some things / state issues detected will temporarily stop for input - the key here is temporarily. They will continue without requiring any action after the temporary timeout (typically 30 seconds).
- \* Full option names are prepended with two dashes, e.g. `--` or `--debug --verbose --ignore-proxy`.
- \* When setting a value to an option, always put an equals (=) between the name and the setting, e.g. `--source=""local""`.
- \* When setting a value to an option, always surround the value properly with double quotes bookending apostrophes, e.g. `--source=""internal\_server""`.
- \* If you are building PowerShell scripts, you can most likely just simply use apostrophes surrounding option values, e.g. `--source='internal\_server'`.
- \* Prefer upgrade to install in scripts. You can't `install` to a newer version of something, but you can `choco upgrade` which will do both upgrade or install (unless switched off explicitly).
- \* If you are sharing the script with others, pass `--source` to be explicit about where the package is coming from. Use full link and not source name ('<https://community.chocolatey.org/api/v2>' versus 'chocolatey').
- \* If parsing output, you might want to use `--limit-output` / '-r' to get output in a more machine parseable format. NOTE: Not all commands handle return of information in an easily digestible output.
- \* Use exit codes to determine status. Chocolatey exits with 0 when everything worked appropriately and other exits codes like 1 when things error. There are package specific exit codes that are recommended to be used and reboot indicating exit codes as well. To check exit code when using PowerShell, immediately call `\$exitCode = \$LASTEXITCODE` to get the value choco exited with.

Here's an example following bad practices (line breaks added for readability):

```
'choco install pkg1 -y -params '/Option:Value /Option2:value with  
spaces' --c4b-option 'Yaass' --option-that-is-new 'dude upgrade"
```

Now here is that example written with best practices (again line breaks added for readability - there are not line continuations

for choco):

```
'choco upgrade pkg1 -y --source="https://community.chocolatey.org/api/v2"  
--package-parameters="/Option:Value /Option2:value with spaces"  
--c4b-option="Yaass" --option-that-is-new="dude upgrade"'
```

Note the differences between the two:

- \* Which is more self-documenting?
- \* Which will allow for the newest version of something installed or upgraded to (which allows for more environmental consistency on packages and versions)?
- \* Which may throw an error on a badly passed option?
- \* Which will throw errors on unknown option values? See explanation below.

Chocolatey ignores options it doesn't understand, but it can only ignore option values if they are tied to the option with an equals sign ('='). Note those last two options in the examples above? If you roll off of a commercial edition or someone with older version attempts to run the badly crafted script `--c4b-option 'Yaass' --option-that-is-new 'dude upgrade'`, they are likely to see errors on 'Yaass' and 'dude upgrade' because they are not explicitly tied to the option they are written after. Now compare that to the other script. Choco will ignore '--c4b-option="Yaass"' and '--option-that-is-new="dude upgrade"' as a whole when it doesn't register the options. This means that your script doesn't error.

Following these scripting best practices will ensure your scripts work everywhere they are used and with newer versions of Chocolatey.

## Default Options and Switches

-?, --help, -h

Prints out the help menu.

--online

Online - Open help for specified command in default browser application.  
This option only works when used in combination with the -?/--help/-h option. Available in 2.0.0+

-d, --debug

Debug - Show debug messaging.

-v, --verbose

Verbose - Show verbose messaging. Very verbose messaging, avoid using under normal circumstances.

--trace

Trace - Show trace messaging. Very, very verbose trace messaging. Avoid except when needing super low-level .NET Framework debugging.

--nocolor, --no-color

No Color - Do not show colorization in logging output. This overrides the feature 'logWithoutColor', set to 'False'.

--acceptlicense, --accept-license

AcceptLicense - Accept license dialogs automatically. Reserved for future use.

**-y, --yes, --confirm**  
Confirm all prompts - Chooses affirmative answer instead of prompting.  
Implies --accept-license

**-f, --force**  
Force - force the behavior. Do not use force during normal operation -  
it subverts some of the smart behavior for commands.

**--noop, --whatif, --what-if**  
NoOp / WhatIf - Don't actually do anything.

**-r, --limitoutput, --limit-output**  
LimitOutput - Limit the output to essential information

**--timeout, --execution-timeout=VALUE**  
CommandExecutionTimeout (in seconds) - The time to allow a command to  
finish before timing out. Overrides the default execution timeout in the  
configuration of 2700 seconds. Supply '0' to disable the timeout.

**-c, --cache, --cachelocation, --cache-location=VALUE**  
CacheLocation - Location for download cache, defaults to %TEMP% or value  
in chocolatey.config file.

**--allowunofficial, --allow-unofficial, --allowunofficialbuild, --allow-unofficial-build**  
AllowUnofficialBuild - When not using the official build you must set  
this flag for choco to continue.

**--failstderr, --failonstderr, --fail-on-stderr, --fail-on-standard-error, --fail-on-error-output**  
FailOnStandardError - Fail on standard error output (stderr), typically  
received when running external commands during install providers. This  
overrides the feature failOnStandardError.

**--use-system-powershell**  
UseSystemPowerShell - Execute PowerShell using an external process  
instead of the built-in PowerShell host. Should only be used when  
internal host is failing.

**--no-progress**  
Do Not Show Progress - Do not show download progress percentages.

**--proxy=VALUE**  
Proxy Location - Explicit proxy location. Overrides the default proxy  
location of "".

**--proxy-user=VALUE**  
Proxy User Name - Explicit proxy user (optional). Requires explicit  
proxy ('--proxy' or config setting). Overrides the default proxy user of  
"".

**--proxy-password=VALUE**  
Proxy Password - Explicit proxy password (optional) to be used with user  
name. Encrypted. Requires explicit proxy ('--proxy' or config setting)  
and user name ('--proxy-user' or config setting). Overrides the default  
proxy password.

**--proxy-bypass-list=VALUE**  
ProxyBypassList - Comma separated list of regex locations to bypass on  
proxy. Requires explicit proxy ('--proxy' or config setting). Overrides  
the default proxy bypass list of "".

--proxy-bypass-on-local  
Proxy Bypass On Local - Bypass proxy for local connections. Requires explicit proxy (`--proxy` or config setting). Overrides the default proxy bypass on local setting of 'True'.

--log-file=VALUE  
Log File to output to in addition to regular loggers.

--skipcompatibilitychecks, --skip-compatibility-checks  
SkipCompatibilityChecks - Prevent warnings being shown before and after command execution when a runtime compatibility problem is found between the version of Chocolatey and the Chocolatey Licensed Extension.

--ignore-http-cache  
IgnoreHttpCache - Ignore any HTTP caches that have previously been created when querying sources, and create new caches. Available in 2.1.0+  
Chocolatey v2.3.0  
PS C:\WINDOWS\system32> node -v  
v20.18.0  
PS C:\WINDOWS\system32> npm -v  
10.8.2  
PS C:\WINDOWS\system32> npm install -g @angular/cli

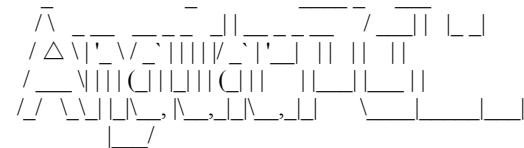
added 266 packages in 41s

49 packages are looking for funding  
run `npm fund` for details  
npm notice  
npm notice New minor version of npm available! 10.8.2 -> 10.9.0  
npm notice Changelog: <https://github.com/npm/cli/releases/tag/v10.9.0>  
npm notice To update run: npm install -g npm@10.9.0  
npm notice  
PS C:\WINDOWS\system32> npm install -g npm@10.9.0

added 1 package in 18s

25 packages are looking for funding  
run `npm fund` for details  
PS C:\WINDOWS\system32> npm fund  
System32

PS C:\WINDOWS\system32> ng v



Angular CLI: 18.2.10  
Node: 20.18.0  
Package Manager: npm 10.9.0  
OS: win32 x64

Angular:

...

Package	Version
---------	---------

```
-----  
@angular-devkit/architect 0.1802.10 (cli-only)  
@angular-devkit/core      18.2.10 (cli-only)  
@angular-devkit/schematics 18.2.10 (cli-only)  
@schematics/angular       18.2.10 (cli-only)
```

```
PS C:\WINDOWS\system32> npm install -g typescript
```

```
added 1 package in 7s
```

```
PS C:\WINDOWS\system32> tsc -v
```

```
Version 5.6.3
```

```
PS C:\WINDOWS\system32> npm install -g @angular/cli
```

```
changed 266 packages in 10s
```

```
49 packages are looking for funding
```

```
  run `npm fund` for details
```

```
PS C:\WINDOWS\system32> npm fund
```

```
System32
```

```
PS C:\WINDOWS\system32> ng --version
```

```
18.2.10
```

```
PS C:\WINDOWS\system32> cd D:\CodePython\
```

```
PS D:\CodePython> mkdir angular
```

```
Directory: D:\CodePython
```

Mode	LastWriteTime	Length	Name
d----	10/24/2024 9:46 AM		angular

```
PS D:\CodePython> cd angular
```

```
PS D:\CodePython\angular> ng new myfirstproject
```

```
Would you like to share pseudonymous usage data about this project with the Angular Team  
at Google under Google's Privacy Policy at https://policies.google.com/privacy. For more  
details and how to change this setting, see https://angular.dev/cli/analytics.
```

```
yes
```

```
Thank you for sharing pseudonymous usage data. Should you change your mind, the following  
command will disable this feature entirely:
```

```
ng analytics disable --global
```

```
Global setting: enabled
```

```
Local setting: No local workspace configuration file.
```

```
Effective status: enabled
```

```
? Which stylesheet format would you like to use? CSS [  
https://developer.mozilla.org/docs/Web/CSS ]
```

```
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? no  
CREATE myfirstproject/angular.json (2718 bytes)
```

```
CREATE myfirstproject/package.json (1086 bytes)
```

```
CREATE myfirstproject/README.md (1103 bytes)
```

```
CREATE myfirstproject/tsconfig.json (1045 bytes)
```

```
CREATE myfirstproject/.editorconfig (331 bytes)
```

```
CREATE myfirstproject/.gitignore (629 bytes)
```

```
CREATE myfirstproject/tsconfig.app.json (439 bytes)
CREATE myfirstproject/tsconfig.spec.json (449 bytes)
CREATE myfirstproject/.vscode/extensions.json (134 bytes)
CREATE myfirstproject/.vscode/launch.json (490 bytes)
CREATE myfirstproject/.vscode/tasks.json (980 bytes)
CREATE myfirstproject/src/main.ts (256 bytes)
CREATE myfirstproject/src/index.html (313 bytes)
CREATE myfirstproject/src/styles.css (81 bytes)
CREATE myfirstproject/src/app/app.component.html (20239 bytes)
CREATE myfirstproject/src/app/app.component.spec.ts (969 bytes)
CREATE myfirstproject/src/app/app.component.ts (323 bytes)
CREATE myfirstproject/src/app/app.component.css (0 bytes)
CREATE myfirstproject/src/app/app.config.ts (318 bytes)
CREATE myfirstproject/src/app/app.routes.ts (80 bytes)
CREATE myfirstproject/public/favicon.ico (15086 bytes)
```

✓ Packages installed successfully.

Successfully initialized git.

```
PS D:\CodePython\angular> cd D:\CodePython\angular\myfirstproject
PS D:\CodePython\angular\myfirstproject> ng serve -o
```

Would you like to share pseudonymous usage data about this project with the Angular Team at Google under Google's Privacy Policy at <https://policies.google.com/privacy>. For more details and how to change this setting, see <https://angular.dev/cli/analytics>.

yes

Thank you for sharing pseudonymous usage data. Should you change your mind, the following command will disable this feature entirely:

```
ng analytics disable
```

Global setting: enabled

Local setting: enabled

Effective status: enabled

Initial chunk files	Names	Raw size
polyfills.js	polyfills	90.20 kB
main.js	main	22.70 kB
styles.css	styles	95 bytes
Initial total   112.99 kB		

Application bundle generation complete. [100.045 seconds]

Watch mode enabled. Watching for file changes...

NOTE: Raw file sizes do not reflect development server per-request transformations.

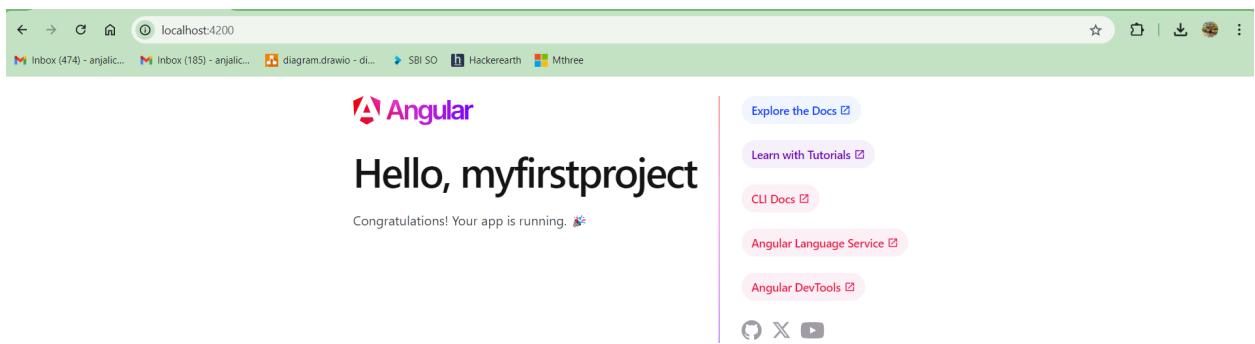
- Local: <http://localhost:4200/>
- press h + enter to show help

---

## **History of Commands:**

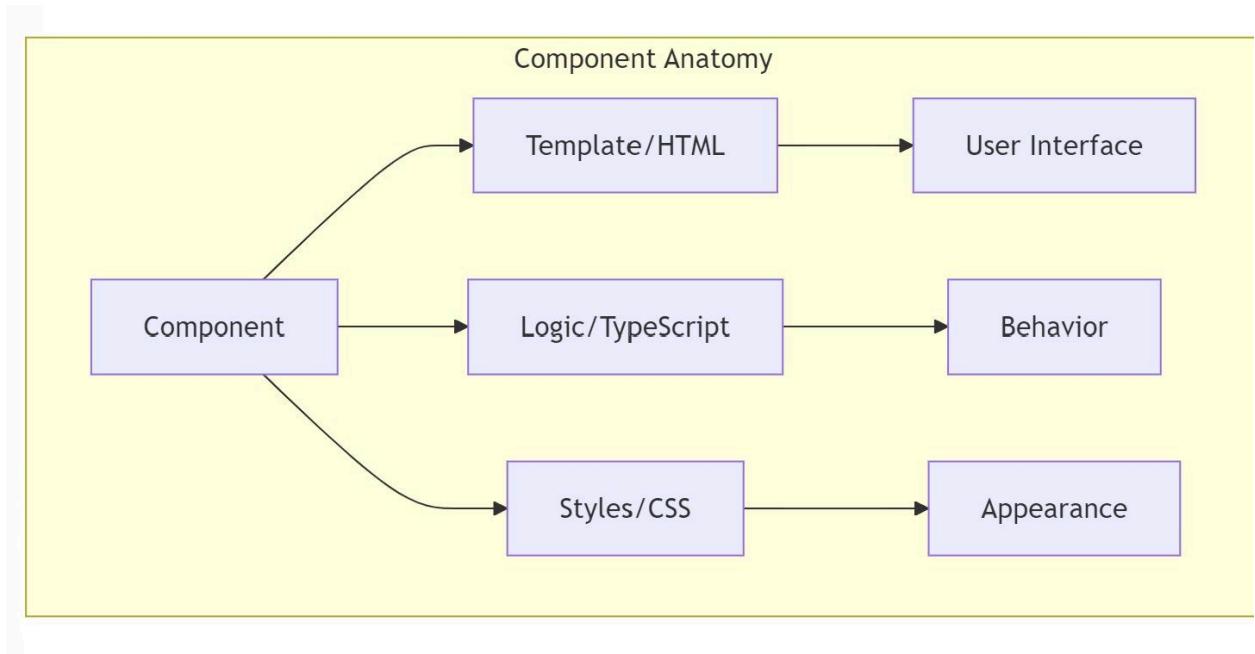
```
PS D:\CodePython\angular\myfirstproject> history
```

Id	CommandLine	-Force;
1	Get -ExecutionPolicy	
2	Get-ExecutionPolicy	
3	Set-ExecutionPolicy AllSigned...	
4	Get-ExecutionPolicy...	
5	Set-ExecutionPolicy Bypass -Scope Process	=
[System.Net.ServicePointManager]::SecurityProtocol		
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Objec...		
6	choco --version...	
7	choco install nodejs-lts --version="20.18.0"	
8	choco -version	
9	node -v	
10	npm -v	
11	npm install -g @angular/cli	
12	npm install -g npm@10.9.0	
13	npm fund	
14	ng v	
15	npm install -g typescript	
16	tsc -v	
17	npm install -g @angular/cli	
18	npm fund	
19	ng --version	
20	cd D:\CodePython\	
21	mkdir angular	
22	cd angular	
23	ng new myfirstproject	
24	cd D:\CodePython\angular\myfirstproject	
25	ng serve -o	



## A "To-Do" application, outlining the flow of adding a new task:

1. User: Enters the text for a new to-do item.
2. Template: The template, likely part of the user interface or the component managing the form, processes the input. It then initiates the process of adding the to-do.
3. Component (Comp): The component responsible for handling to-dos creates a new to-do item based on the input text.
4. Store (State Management): A new to-do item is created and added to the store or state, which holds the list of to-dos.
5. Component: After the to-do is added, the UI is updated to reflect the new item in the list of to-dos.



---

## A TypeScript/JavaScript-like structure, possibly for an Angular component:

```
selector: 'app-tv',  
  
// Screen - What viewers see  
template: `'  
  <div>  
    <h1>Channel: {{ currentChannel }}</h1>  
    <button (click)="changeChannel()">Change Channel</button>  
  </div>  
`,  
  
styles: [  
  div {  
    background: black;  
  }  
  h1 {  
    color: white;  
  }  
,  
]  
  
class TVComponent {  
  currentChannel = 'Some Channel';  
  
  changeChannel() {  
    // Logic to change channel goes here  
  }  
}
```

Explanation:

1. Component selector: `app-tv` - This defines how this component will be used in HTML, `<app-tv></app-tv>`.
2. Template: This is the HTML that users will see. It displays the current channel and a button to change the channel.
3. Binding and event handling:
  - `{{ currentChannel }}`: Interpolates the `currentChannel` value into the template.
  - `(click)="changeChannel()"`: Calls the `changeChannel()` method when the button is clicked.
4. Styles: The background of the `div` is black, and the text color of the `h1` element is white.

## An Angular component, involving event binding:

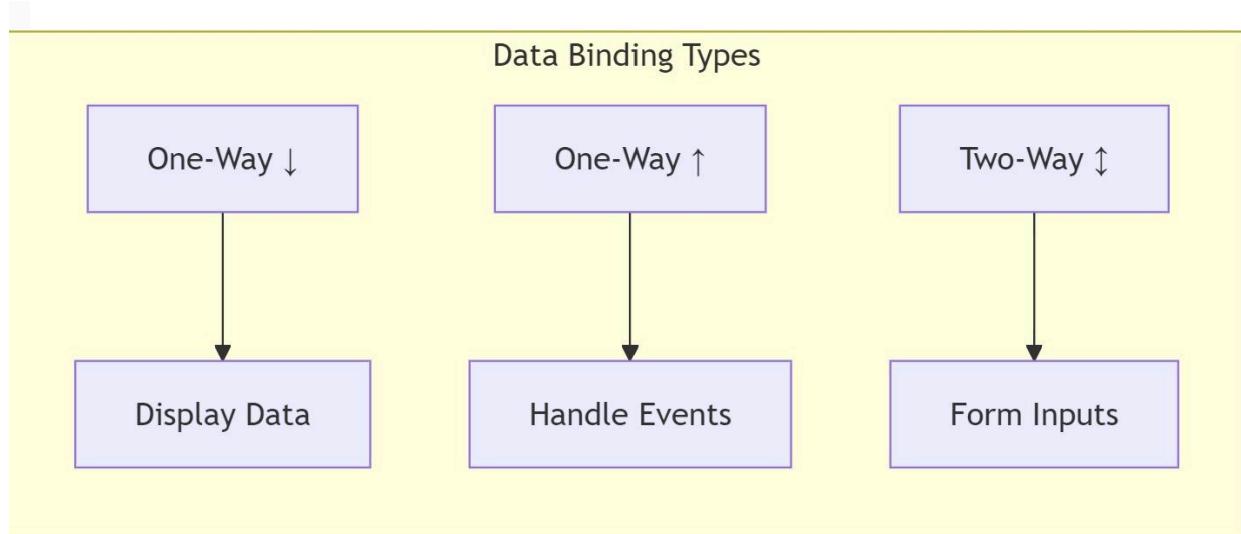
```
@Component({  
  template: `  
    <button (click)="callWaiter()">  
      Need Help  
    </button>  
  `  
})  
class TableComponent {  
  callWaiter() {  
    console.log("Waiter called");  
  }  
}
```

### Key Details:

1. Component template: The template contains a button that says "Need Help." When clicked, it triggers the `callWaiter()` function.
2. Event binding: The `(click)` event is bound to the `callWaiter()` method.
3. Method: The `callWaiter()` method logs the message "Waiter called" to the console.

### Concepts:

- Event binding: This is Angular's way of binding events in the view to methods in the component, as shown by the `(click)` event.
- Component: In Angular, a component contains both the view (template) and logic (methods like `callWaiter()`).



---

**Like a restaurant menu board showing prices:**

```
@Component({
  template: `

    <h1>Today's Special: {{dishName}}</h1>
    <p>Price: ${{price}}</p>

`)

class MenuComponent {
  dishName = "Pizza";
  price = 10;
}
```

**Like pressing a button to call waiter:**

```
@Component({
  template: `

    <button (click)="callWaiter()">Need Help</button>

`)

class TableComponent {
  callWaiter() {
    console.log("Waiter called!");
  }
}
```

**Hotel Service Example:**

```
@Injectable({
  providedIn: 'root'
})

class HotelService {
  // Shared resources (like cleaning supplies)
  private rooms = [];

  // Shared functions (like cleaning procedures)
  cleanRoom(roomNumber: number) {
    console.log(`Cleaning room ${roomNumber}`);
  }

  // Data management (like room status)
  getRoomStatus(roomNumber: number) {
    return this.rooms[roomNumber];
  }
}
```

```
// Using the service in a component (like a floor manager)
@Component({})
class FloorComponent {
  constructor(private hotelService: HotelService) {
    // Can now use cleaning service
    this.hotelService.cleanRoom(101);
  }
}
```

#### **Like instructions: "If box is heavy, get help"**

```
@Component({
  template: `
    <div *ngIf="isHeavy">
      Please get assistance!
    </div>

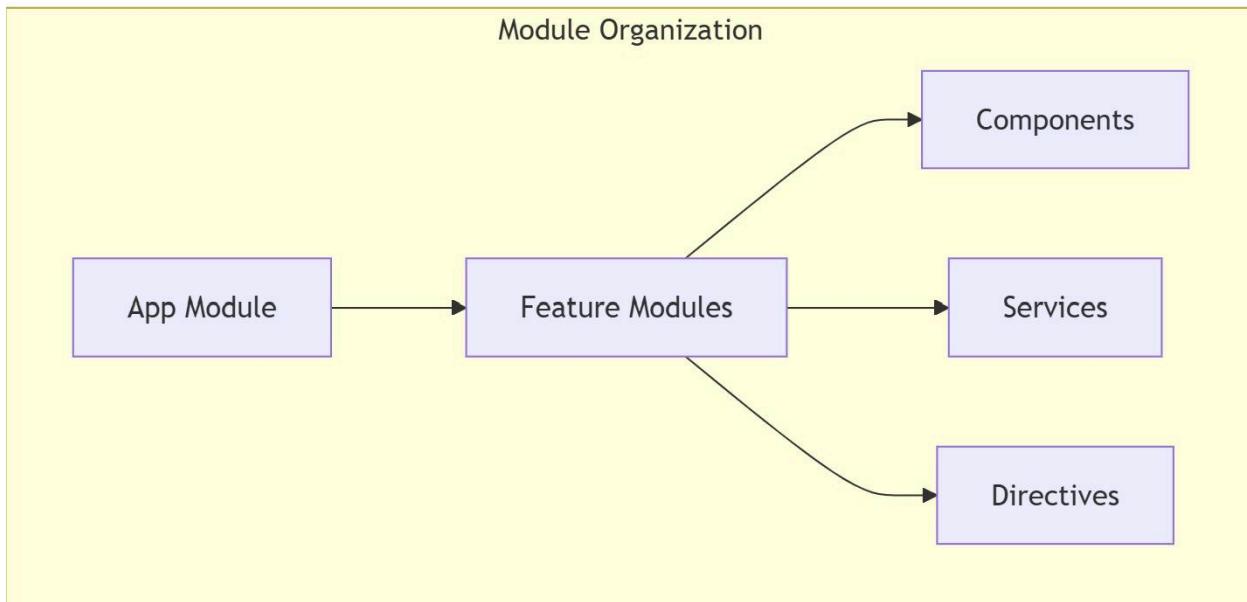
    <!-- Like "Repeat for each screw" -->
    <div *ngFor="let item of parts">
      Install {{item}}
    </div>
  `
})
```

#### **Like "Paint this part red if it's important"**

```
@Component({
  template: `
    <div [ngStyle]="{'color': isImportant ? 'red' : 'black'}">
      Important Note
    </div>
  `
})
```

## Day 29 - Angular Continues...

### Angular Modules:



Declarations: `[ ] // Components, Directives`

- This array holds the components, directives, and pipes that belong to the module.

Imports: `[ ] // Other modules`

- This array imports other modules whose exported classes are needed by the components in this module.

Exports: `[ ] // What you want to share`

- This array defines the subset of declarations that should be visible and usable in the component templates of other modules.

Providers: `[ ] // Services`

- This array lists the services (like Angular services) that will be available across the module.

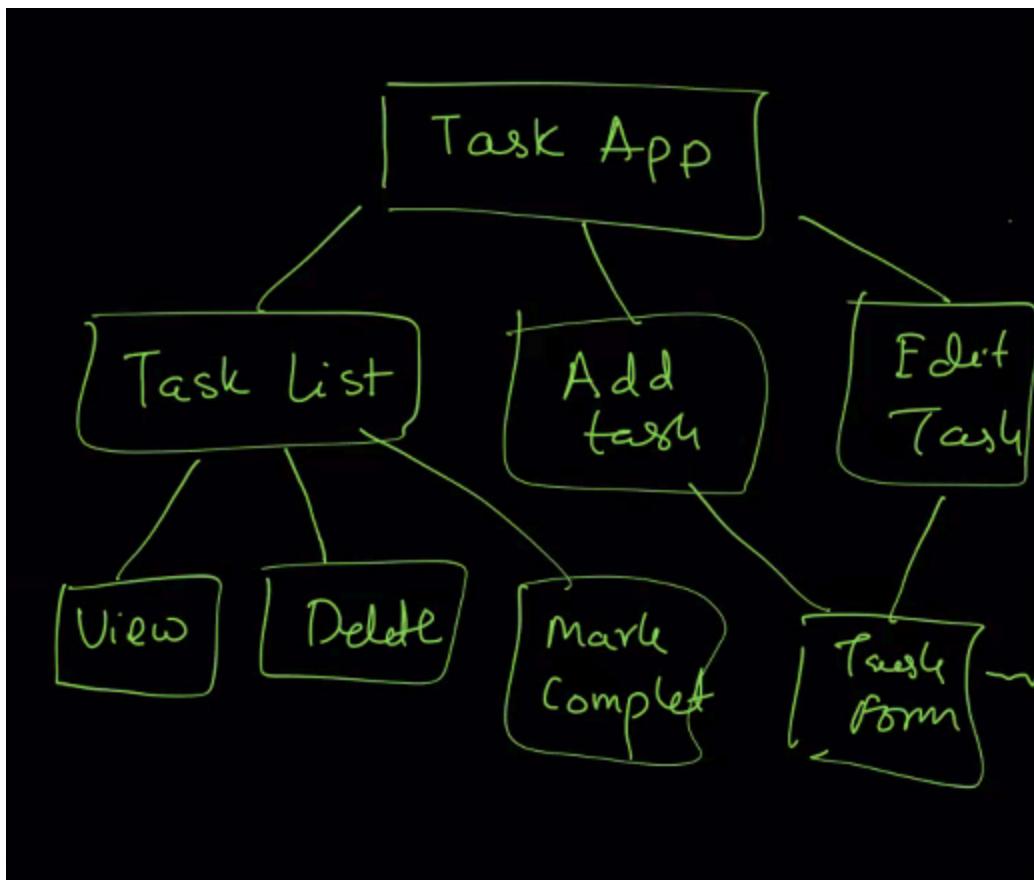
Bootstrap: [ ] // Root component

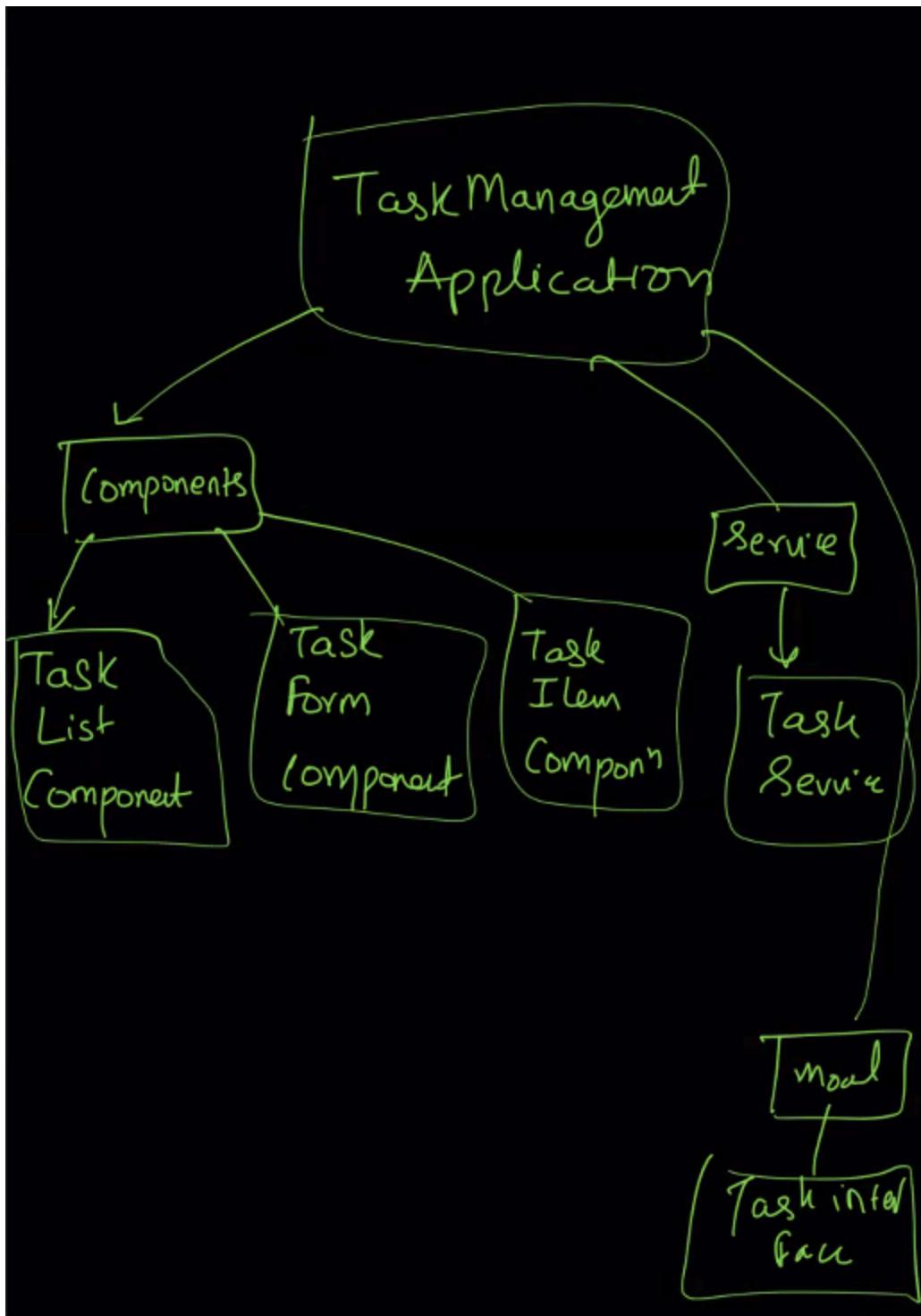
- This array is used in the root module to specify the main component that Angular should bootstrap when the application starts.

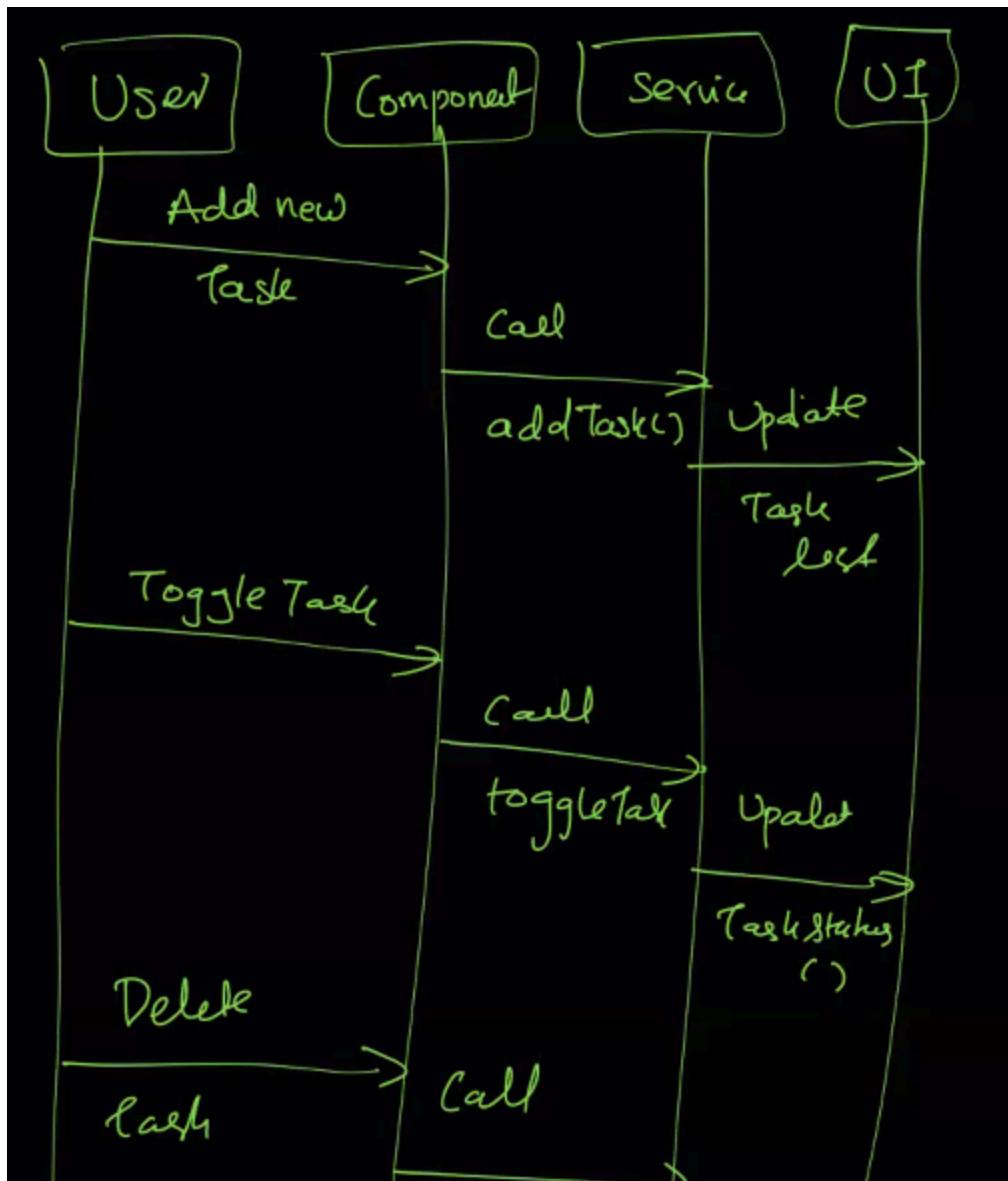
```
export class FeatureModule {  
}
```

- This appears to be an Angular module class being exported, likely as part of defining a feature module.

### Task Application using Angular



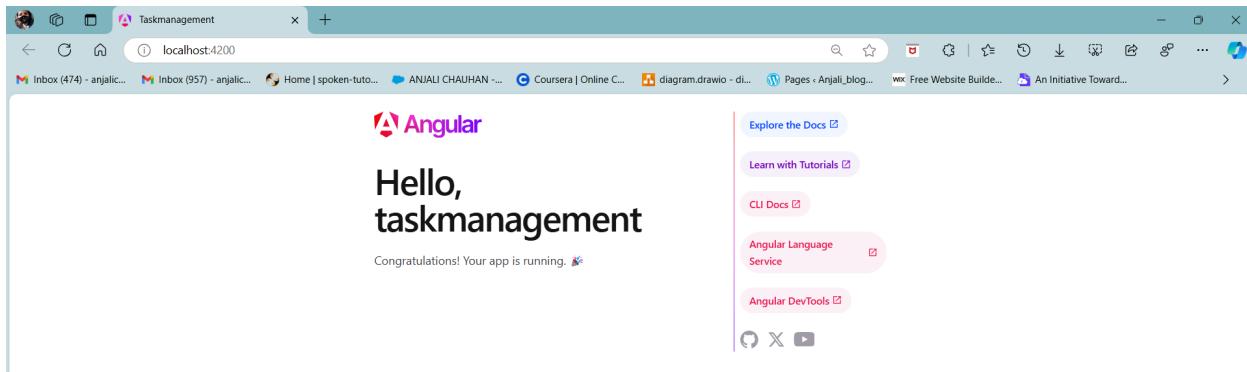




---

## **History of Commands:**

1. 52 cd secondproject
2. 53 cd D:\CodePython
3. 54 cd secondproject
4. 55 ng new taskmanagement
5. 56 cd..
6. 57 cd angular
7. 58 mkdir secondproject
8. 59 ng new taskmanagement
9. 60 npm config list -g
10. 61 npm config delete https-proxy -g
11. 62 ng new taskmanagement
12. 63 npm config get https-proxy
13. 64 npm config delete https-proxy
14. 65 npm config list -g...
15. 66 ng new taskmanagement...
16. 67 ls -lrt
17. 68 ls
18. 69 cd .\taskmanagement\
19. 70 ng serve -o
20. 71 ls
21. 72 ng generate component components/task-list...
22. 73 ng generate component components/task-form...
23. 74 ng generate component components/task-item...
24. 75 ng generate service services/task



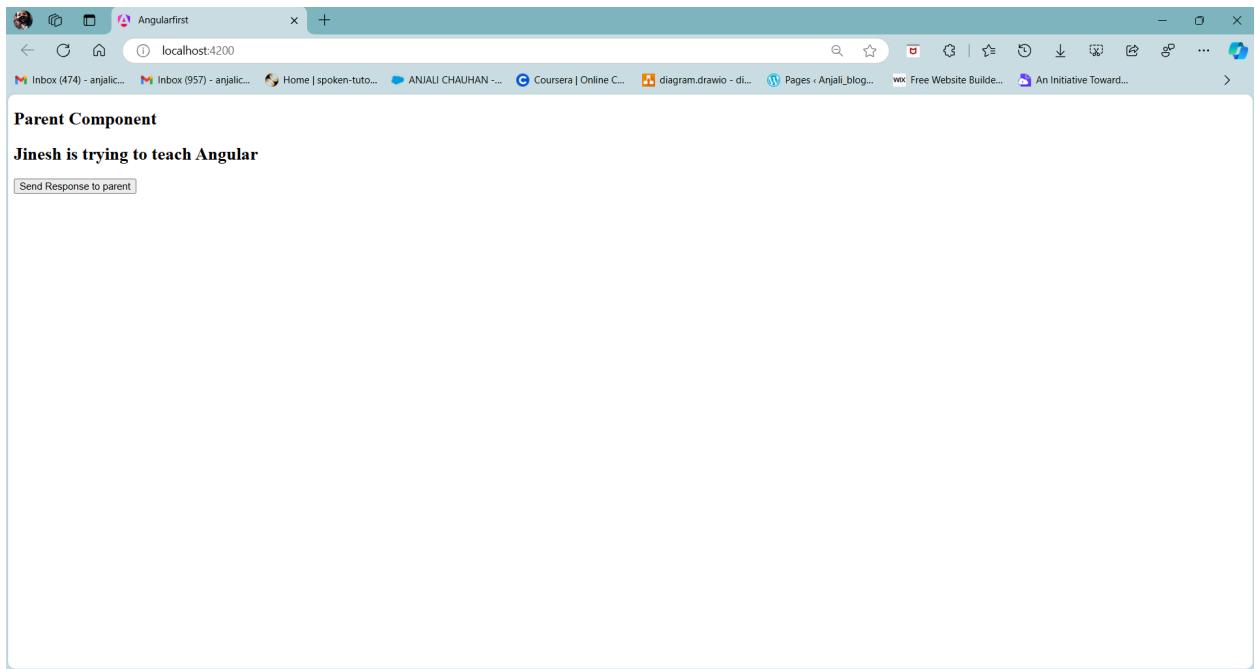
## Basic Angular Project:

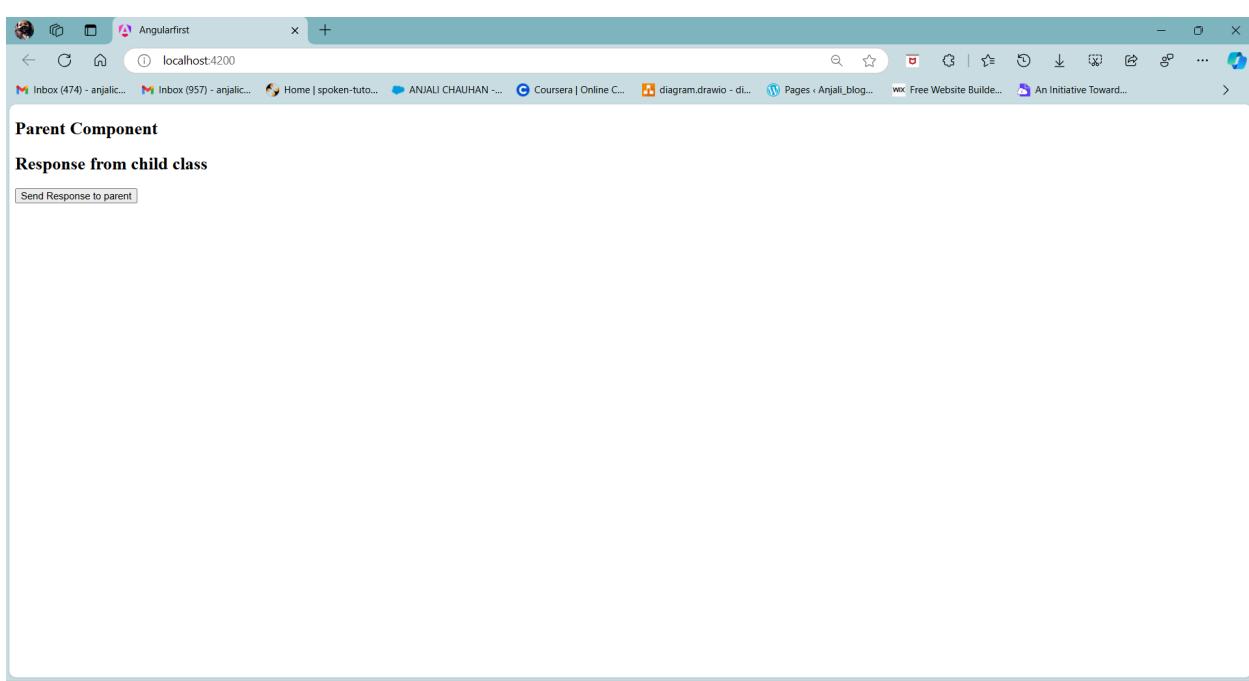
```
src > app > # app.component.css > .task-container li
1   .task-container {
2     background-color: #f0f0f0;
3     padding: 20px;
4     border-radius: 10px;
5     margin: 20px;
6   }
7
8   .task-container ul {
9     list-style-type: none;
10    padding: 0;
11  }
12
13  .task-container li {
14    margin-bottom: 10px;
15    border-bottom: 1px solid #ccc;
16 }
```

```
src > app > app.component.html > div
1   <div>
2     <h2>Parent Component</h2>
3     <app-child [message]="parentMessage" (response)="handleResponse($event)">
4     </app-child>
5   </div>
```

```
src > app > app.component.ts > AppComponent
1   import { Component } from '@angular/core';
2   import { CommonModule } from '@angular/common';
3   import { ChildComponent } from './child.component';
4   import { RouterOutlet } from '@angular/router';
5
6   @Component({
7     selector: 'app-root',
8     standalone: true,
9     imports: [CommonModule, RouterOutlet, ChildComponent],
10    templateUrl: './app.component.html'
11  })
12  export class AppComponent {
13    parentMessage = 'Jinesh is trying to teach Angular';
14
15    handleResponse(response:string){
16      this.parentMessage = response;
17    }
18  }
19
```

```
src > app > ts child.component.ts > ChildComponent
1 import { Component, Input, Output, EventEmitter } from '@angular/core';
2
3 @Component({
4   selector: 'app-child',
5   standalone: true,
6   template: `<div><h2>{{message}}</h2> <button (click)="sendResponse()">Send Response to parent</button>
7 })
8 export class ChildComponent {
9   @Input() message: string = '';
10  @Output() response = new EventEmitter<string>();
11
12  sendResponse(){
13    this.response.emit('Response from child class');
14  }
15}
16
```





```
PS C:\WINDOWS\system32> cd D:\CodePython\
PS D:\CodePython> cd angular
PS D:\CodePython\angular> cd basicangular
PS D:\CodePython\angular\basicangular> cd .\angularfirst\
PS D:\CodePython\angular\basicangular\angularfirst> ng serve -o
Initial chunk files | Names      | Raw size
polyfills.js        | polyfills  | 90.20 kB |
main.js            | main       | 3.75 kB |
styles.css         | styles     | 95 bytes |

| Initial total | 94.05 kB
```

Application bundle generation complete. [14.284 seconds]

Watch mode enabled. Watching for file changes...

NOTE: Raw file sizes do not reflect development server per-request transformations.

- ➔ Local: http://localhost:4200/
- ➔ press h + enter to show help

## Day 30 - Angular Applications

Component Structure:

The `@Component` decorator defines the component's metadata  
`standalone: true` makes it a standalone component  
`selector` defines how to use the component in HTML (`<todo-app>`)  
`template` contains the component's HTML structure  
`styles` contains component-specific CSS

State Management:

Component state is managed through class properties (`todos`, `newTodoText`, `nextId`)  
TypeScript interface (`Todo`) ensures type safety  
State is modified through class methods

Template Features:

Data binding:

`{{ }}` for displaying values  
`[]` for property binding  
`()` for event binding

`@for` directive for iterating over lists  
Conditional styling with `[class.completed]`

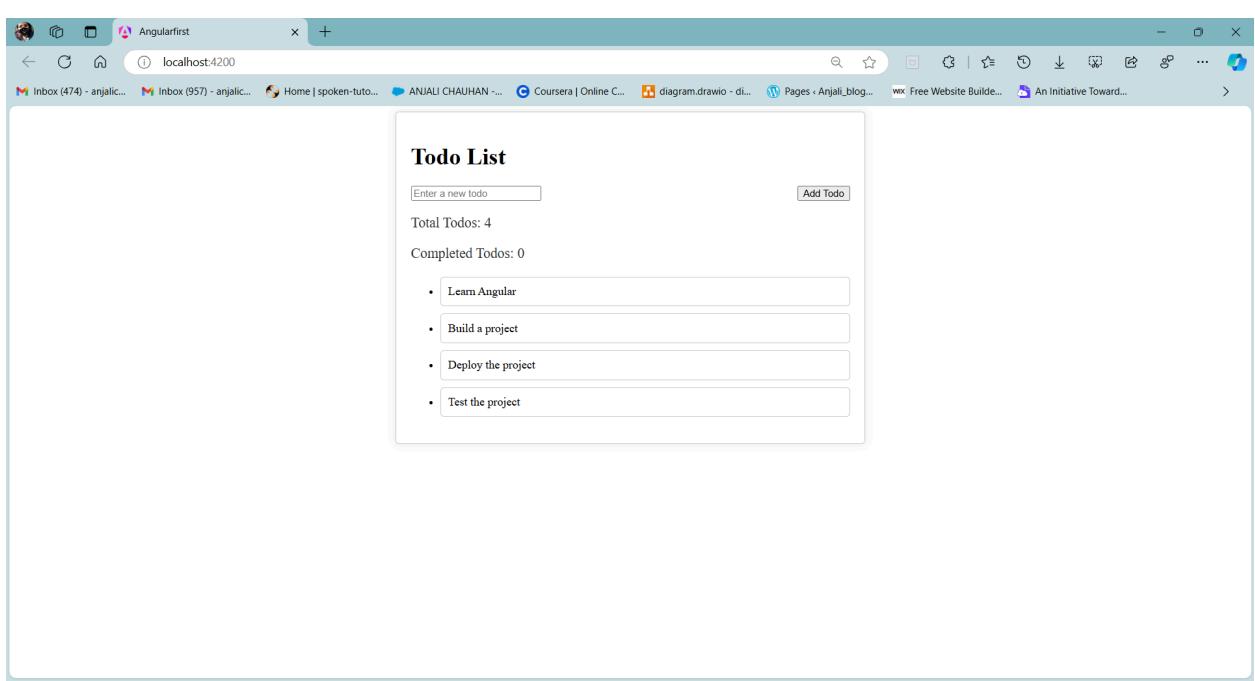
User Interaction:

Input handling with event binding  
Click handlers for buttons  
Checkbox state management

## To-Do Application

```
src > app > ts app/components > AppComponent.ts
1 import { Component, OnInit, OnDestroy } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4 interface Todo{
5   id:number;
6   text:string;
7   completed:boolean;
8 }
9 @Component({
10   selector: 'app-root',
11   standalone: true,
12   imports: [CommonModule],
13   template: `<div class="todo-container">
14 <h1>Todo List</h1>
15
16 <!--Form to add new todo-->
17 <div class="add-todo">
18   <input type="text" [value]="newTodoText" (input)="updateNewTodo($event)" placeholder="Enter a new todo " />
19   <button (click)="addTodo()">Add Todo</button>
20 </div>
21
22 <!-- Statistics section-->
23 <div class="statistics">
24   <p>Total Todos: {{todos.length}}</p>
25   <p>Completed Todos: {{getCompletedTodosCount()}}</p>
26 </div>
27
28 <!--Todo List section-->
29 <div class="todo-list">
30   <ul>
31     <li *ngFor="let todo of todos">{{todo.text}}</li>
32   </ul>
33 </div>
34 `,
35 styles: ` .todo-container{
36   max-width: 600px;
37   margin: 0 auto;
38   padding: 20px;
39   border: 1px solid #ccc;
40   border-radius: 5px;
41   box-shadow: 0 0 10px rgba(0,0,0,0.1);
42 }
43 .add-todo{
44   display: flex;
45   justify-content: space-between;
46   margin-bottom: 20px;
47 }
48 .todo-list{
49   margin-top: 20px;
50 }
51 .statistics{
52   margin-top: 20px;
53   font-size: 1.2em;
54   color: #333;
55 }
```

```
56  .todo-list li{
57    margin-bottom: 10px;
58    padding: 10px;
59    border: 1px solid #ccc;
60    border-radius: 5px;
61  }
62  .todo-list li.completed{
63    background-color: #f0f0f0;
64    color: #888;
65  }
66 }
67 export class AppComponent {
68
69
70 todos:Todo[] = [
71   {id:1,text:"Learn Angular",completed:false},
72   {id:2,text:"Build a project",completed:false},
73   {id:3,text:"Deploy the project",completed:false}
74 ];
75
76 newTodoText:string = '';
77 nextId:number = 4;
78
79 updateNewTodo(event:Event):void{
80   const input =event.target as HTMLInputElement;
81   this.newTodoText = input.value;
82 }
83
84 addTodo():void{
85   if(this.newTodoText.trim() === ''){
86     return;
87   }
88   this.todos.push({id:this.nextId++,text:this.newTodoText,completed:false});
89   this.newTodoText = '';
90 }
91
92
93 getCompletedTodosCount():number{
94   return this.todos.filter(todo => todo.completed).length;
95 }
96 }
```

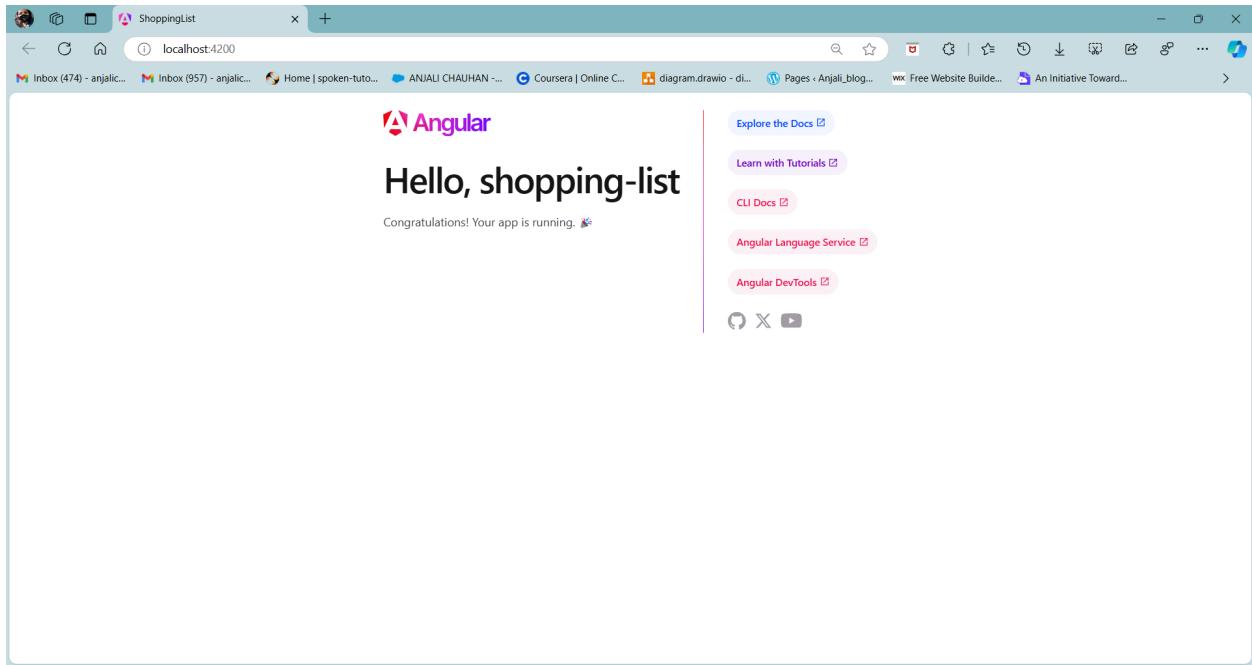


## Shopping - List Application

```
ng new shopping-list
```

```
cd shopping-list
```

```
ng serve -o
```



# Day 31 - Shopping List Application and Project

```
ts shopping.service.ts | ts shopping-list.component.ts | ts shopping-list.types.ts
angular > basicangular > shopping-list-app > src > app > services > ts shopping.service.ts > ...
1   Ctrl+L to chat, Ctrl+K to generate
2   import { HttpClient } from '@angular/common/http';
3   import { Observable } from 'rxjs';
4   import { ShoppingItem } from '../app.component';
5
6
7   import { Injectable } from '@angular/core';
8
9   @Injectable({
10     providedIn: 'root'
11   })
12   export class ShoppingService {
13     private apiUrl = "http://localhost:8080/api/shopping";
14
15     constructor(private http: HttpClient) {}
16
17     getItems(): Observable<ShoppingItem[]> {
18       return this.http.get<ShoppingItem[]>(`${this.apiUrl}/items`); // Fixed with backticks
19     }
20
21     getItemsByCategory(category: string): Observable<ShoppingItem[]> {
22       return this.http.get<ShoppingItem[]>(`${this.apiUrl}/items/category/${category}`); // Fixed with backticks
23     }
24
25     // Post new item
26     addItem(item: Omit<ShoppingItem, 'id'>): Observable<ShoppingItem> {
27       return this.http.post<ShoppingItem>(`${this.apiUrl}/items`, item); // Fixed with backticks
28     }
29
30     // PUT update item
31     updateItem(item: ShoppingItem): Observable<ShoppingItem> {
32       return this.http.put<ShoppingItem>(`${this.apiUrl}/items/${item.id}`, item); // Fixed with backticks
33     }
34
35     // DELETE item
36     deleteItem(id: number): Observable<void> {
37       return this.http.delete<void>(`${this.apiUrl}/items/${id}`); // Fixed with backticks
38     }
39
40     // Update purchased status
41     updatePurchasedStatus(id: number, isPurchased: boolean): Observable<ShoppingItem> {
42       return this.http.patch<ShoppingItem>(`${this.apiUrl}/items/${id}/purchasedStatus`, { isPurchased }); // Fixed with backticks
43     }
44
45 }
```

```

angular > basicangular > shopping-list-app > src > app > shopping-list > shopping-list.component.ts ...
1 import { Component } from '@angular/core';
2 import { ShoppingItem } from './shopping-list.types';
3 import { FormsModule } from '@angular/forms';
4
5 @Component({
6   selector: 'shopping-list',
7   standalone: true,
8   imports: [FormsModule],
9   template: '<div class="shopping-container">
10   <h1>Shopping List</h1>
11   <!-- Add your form and shopping list structure here -->
12   </div>',
13   styles: [
14     /* Add your component styles here */
15   ]
16 })
17 export class ShoppingItemComponent {
18   items: ShoppingItem[] = [
19     { id: 1, name: 'Milk', quantity: 2, category: 'Groceries', isPurchased: false },
20     { id: 2, name: 'Bread', quantity: 1, category: 'Groceries', isPurchased: false },
21     { id: 3, name: 'Eggs', quantity: 12, category: 'Groceries', isPurchased: false },
22   ];
23
24   isAddItem: boolean = false;
25   newItem: Partial<ShoppingItem> = {
26     name: '',
27     quantity: 1,
28     category: 'Groceries',
29     isPurchased: false,
30   };
31
32   editingItemId: number | null = null;
33   editingItem: Partial<ShoppingItem> = {};
34   selectedCategory: string = 'All';
35   private nextItemId: number = 4;
36
37   startAddingItem() {
38     this.isAddItem = true;
39   }
40
41   cancelAddingItem() {
42     this.isAddItem = false;
43     this.resetNewItem();
44   }
45
46   resetNewItem() {
47     this.newItem = {
48       name: '',
49       quantity: 1,
50       category: 'Groceries',
51       isPurchased: false,
52     };
53   }
54
55   addItem() {
56     if(!this.newItem.name || !this.newItem.quantity || !this.newItem.category) {
57       return;
58     }
59     this.items.push({
60       id: this.nextItemId++,
61       name: this.newItem.name,
62       quantity: this.newItem.quantity,
63       category: this.newItem.category,
64       isPurchased: this.newItem.isPurchased ?? false
65     });
66     this.isAddItem = false;
67     this.resetNewItem();
68   }
69
70   startEditingItem(item: ShoppingItem) {
71     this.editingItemId = item.id;
72     this.editingItem = {...item};
73   }
74
75   getPurchasedItemsCount() {
76     return this.items.filter(item => item.isPurchased).length;
77   }
78 }
79

```

```
angular@basicangular > shopping-list-app > src > app > shopping-list > shopping-list.types.ts > ShoppingItem
1 export interface ShoppingItem {
2   id: number;
3   name: string;
4   quantity: number;
5   category: 'Groceries' | 'Household' | 'Electronics' | 'Clothing';
6   isPurchased: boolean;
7 }
8
```

```
TS shopping.service.ts | TS shopping-list.component.ts | TS shopping-list.types.ts | # app.component.css | app.component.html | TS main.server.ts X
angular@basicangular > shopping-list-app > src > main.server.ts > ...
1 import { bootstrapApplication } from '@angular/platform-browser';
2 import { AppComponent } from './app/app.component';
3 import { config } from './app/app.config.server';
4
5 const bootstrap = () => bootstrapApplication(AppComponent, config);
6
7 export default bootstrap;
8
```