# DAY 1 (16 September)

**Software Development Life Cycle (SDLC):**
SDLC is a systematic approach to software development, ensuring it aligns with user needs and maintains high quality.

**The are mainly 6 phases of SDLC:**

- **Requirements Gathering:** Determining the objectives and constraints of the software project.
- **Design:** Creating detailed plans for system architecture and user interfaces.
- **Development:** Implementing the software through coding.
- **Testing:** Verifying that the software functions correctly and meets requirements.
- **Deployment:** Launching the software for end-users.
- **Maintenance:** Performing updates and resolving issues post-deployment.

**Types of Requirements:**

- **Business Requirements:** Strategic goals from a business perspective.
- **User Requirements:** Detailed needs and functionalities required by users.
- **System Requirements:** Specific technical criteria that must be fulfilled (e.g., hardware specs, performance benchmarks).

**SDLC Models:**

- **Waterfall Model:** A sequential design process where each phase is completed before the next begins. It's a traditional method with limited flexibility for changes.
- **Iterative Model:** Develops software in repeated cycles (iterations), allowing for continual feedback and improvements.
- **Prototype Model:** Early development of a prototype to gather user feedback and refine the final system.
- **Spiral Model:** Integrates iterative development with risk management through repetitive cycles of planning, designing, and testing.
- **V-Shape Model:** Stresses validation and verification by matching each development stage with a corresponding testing phase.

**Agile Methodology:**

- Agile focuses on iterative development with frequent revisions and feedback. Development is divided into short cycles called sprints (typically 1-2 weeks).
- Emphasizes adaptability, ongoing stakeholder engagement, and incremental delivery of software components.

**Epics, Features, Stories, and Tasks:**

- **Epic:** A broad goal or large-scale project that is divided into smaller, manageable features.
- **Feature:** A significant function or aspect of the epic that can be developed within a short time frame.
- **Story:** A user-focused requirement that delivers incremental value and can be completed in a few days.
- **Task:** The smallest unit of work, usually assigned to a single individual, and should be achievable within 4 hours.

**GitHub Basics:**

- GitHub facilitates version control and team collaboration through Git.
- Create and manage repositories, monitor changes, and collaborate on code efficiently.
- Utilize project boards to manage tasks and track progress with lists such as To Do, In Progress, and Done.

**Trello & Jira (Project Management Tools):**

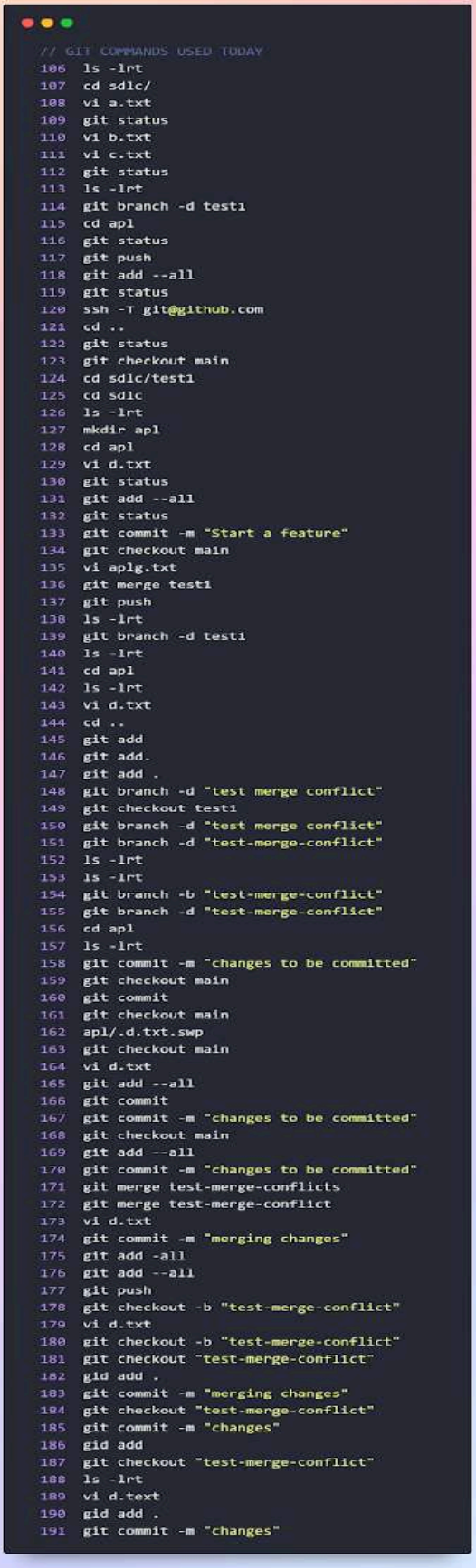
- **Trello:** An intuitive tool for visual task management using boards, lists, and cards. Tasks can be organized with deadlines and responsible individuals.
- **Jira:** A comprehensive tool for Agile project management, used to assign tasks, create sub-tasks, and monitor progress.

## Git Commands used:

```
  // GIT COMMANDS USED TODAY
cd test123sdlc/
ls -lrt
git status
vi readmetest.md
vi readmetest.md
git status
git branch test1
git checkout test1
git branch test2
git branch -a
git branch -d test2
git branch -a
ls -lrt
git status
git status
git add readmetest.md
git status
git commit -m "Added new file for test readme"
git push
git push --set-upstream origin test1
vi a.txt
vi b.txt
git add --all
git commit -m "Added new file for test readme"
git push --set-upstream origin test1
```

# DAY 2 (17 September)

## Git Commands used:



```
// GIT COMMANDS USED TODAY
ls -lrt
cd sdlc/
vi a.txt
git status
vi b.txt
vi c.txt
git status
ls -lrt
git branch -d test1
cd apl
git status
git push
git add --all
git status
ssh -T git@github.com
cd ..
git status
git checkout main
cd sdlc/test1
cd sdlc
ls -lrt
mkdir apl
cd apl
vi d.txt
git status
git add --all
git status
git commit -m "Start a feature"
git checkout main
vi aplg.txt
git merge test1
git push
ls -lrt
git branch -d test1
ls -lrt
cd apl
ls -lrt
vi d.txt
cd ..
git add
git add.
git add .
git branch -d "test merge conflict"
git checkout test1
git branch -d "test merge conflict"
git branch -d "test-merge-conflict"
ls -lrt
ls -lrt
git branch -b "test-merge-conflict"
git branch -d "test-merge-conflict"
cd apl
ls -lrt
git commit -m "changes to be committed"
git checkout main
git commit
git checkout main
apl/.d.txt.swp
git checkout main
vi d.txt
git add --all
git commit
git commit -m "changes to be committed"
git checkout main
git add --all
git commit -m "changes to be committed"
git merge test-merge-conflicts
git merge test-merge-conflict
vi d.txt
git commit -m "merging changes"
git add -all
git add --all
git push
git checkout -b "test-merge-conflict"
vi d.txt
git checkout -b "test-merge-conflict"
git checkout "test-merge-conflict"
gid add .
git commit -m "merging changes"
git checkout "test-merge-conflict"
git commit -m "changes"
gid add
git checkout "test-merge-conflict"
ls -lrt
vi d.text
gid add .
git commit -m "changes"
```

```
// GIT COMMANDS USED TODAY
192  git checkout main
193  git merge test-merge-conflict
194  git log --merge
195  git diff
196  git add
197  gid 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"> 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.supporting arg"> 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 git_tutorial
237  cd git_tutorial/
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
```

# DAY 3 (18 September)

```java
// CODE 1

package main.java;

public class Employee {
    public String name;
    public int age;
    public String city;
    public double salary;
    public String companyName;
    public String employeeId;
    public String department;
    public String designation;

    public void printDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("City: " + city);
        System.out.println("Salary: " + salary);
        System.out.println("Company Name: " + companyName);
        System.out.println("Employee ID: " + employeeId);
        System.out.println("Department: " + department);
        System.out.println("Designation: " + designation);
    }

    public void setDetails(String name, int age, String city, double salary, String companyName, String employeeId, String department, String designation) {
        this.name = name;
        this.age = age;
        this.city = city;
        this.salary = salary;
        this.companyName = companyName;
        this.employeeId = employeeId;
        this.department = department;
        this.designation = designation;
    }

    public Employee() {
        name = "Alice Smith";
        age = 28;
        city = "San Francisco";
        salary = 90000;
        companyName = "XYZ Ltd.";
        employeeId = "9876543210";
        department = "Marketing";
        designation = "Marketing Manager";
        printDetails();
    }

    public Employee(String name, int age, String city, double salary, String companyName, String employeeId, String department, String designation) {
        this.name = name;
        this.age = age;
        this.city = city;
        this.salary = salary;
        this.companyName = companyName;
        this.employeeId = employeeId;
        this.department = department;
        this.designation = designation;
        printDetails();
    }

    public static void main(String[] args) {
        Employee obj = new Employee();
        obj.setDetails("Emily", 29, "Los Angeles", 95000, "DEF Corp.", "1122334455", "HR", "HR Specialist");
        obj.printDetails();
        Employee obj2 = new Employee("Michael", 35, "Seattle", 110000, "GHI Inc.", "2233445566", "Finance", "Financial Analyst");
        obj2.printDetails();
    }
}
```

```java
// CODE 2

package main.java;
//Code to cover static keyword concept in Java
public class staticExample{
    private static int count =0;

    private int instanceNumber;

    public staticExample(){
        count++;
        instanceNumber = count;
    }

    public void display(){
        System.out.println("Instance Number: " + instanceNumber+"\n"+"Count: "+count);
    }

    public static void main(String[] args) {
        staticExample obj1 = new staticExample();
        obj1.display();
        staticExample obj2 = new staticExample();
        obj2.display();
        staticExample obj3 = new staticExample();


        obj3.display();
    }
}
```

```java
// CODE 3

public class TypeCastingExample {
    public static void main(String[] args) {
        // Starting with byte
        byte smallValue = 15;
        System.out.println("Original byte value: " + smallValue);

        // byte to short
        short mediumValue = smallValue;
        System.out.println("byte to short: " + mediumValue);

        // short to char
        // Note: This is not a direct widening conversion, as char is unsigned
        // We'll use a positive short value to demonstrate
        short positiveShortValue = 66;  // ASCII value for 'B'
        char characterValue = (char) positiveShortValue;
        System.out.println("short to char: " + characterValue);

        // char to int
        int integerFromChar = characterValue;
        System.out.println("char to int: " + integerFromChar);

        // int to long
        long longValue = integerFromChar;
        System.out.println("int to long: " + longValue);

        // long to float
        float floatValue = longValue;
        System.out.println("long to float: " + floatValue);

        // float to double
        double doubleValue = floatValue;
        System.out.println("float to double: " + doubleValue);

        // Demonstrating multiple steps of widening in one assignment
        int initialInt = 654321;
        double doubleFromInt = initialInt;
        System.out.println("int directly to double: " + doubleFromInt);

        // Demonstrating widening in expressions
        byte smallNumber = 20;
        short mediumNumber = 200;
        int sumResult = smallNumber + mediumNumber;  // byte and short are promoted to int
        System.out.println("Result of byte + short (as int): " + sumResult);

        // Widening with literals
        int largeInteger = 1_500_000_000;  // 1.5 billion
        long hugeNumber = largeInteger * 4L;  // Result is too large for int, so it's widened to long
        System.out.println("Large calculation result (as long): " + hugeNumber);

        // Demonstrating potential loss of precision
        long veryLargeLong = 9876543219876543211;
        float approximateFloat = veryLargeLong;  // Potential loss of precision
        System.out.println("Long to float (potential precision loss): " + approximateFloat);
    }
}
```

```java
// CODE 4

public class SimpleLogicalOperatorsDemo {
    public static void main(String[] args) {
        // Boolean variables for demonstration
        boolean t3 = true;
        boolean f3 = false;

        // Basic Logical Operations
        System.out.println("1. Basic Logical Operations:");
        System.out.println("   AND: true && true = " + (t3 && t3));
        System.out.println("   AND: true && false = " + (t3 && f3));
        System.out.println("   AND: false && true = " + (f3 && t3));
        System.out.println("   AND: false && false = " + (f3 && f3));
        System.out.println("   OR: true || true = " + (t3 || t3));
        System.out.println("   OR: true || false = " + (t3 || f3));
        System.out.println("   OR: false || true = " + (f3 || t3));
        System.out.println("   OR: false || false = " + (f3 || f3));
        System.out.println("   NOT: !true = " + (!t3));
        System.out.println("   NOT: !false = " + (!f3));

        // Short-circuit Evaluation
        System.out.println("\n2. Short-circuit Evaluation:");
        System.out.println("   false && (1/0 > 0) = " + (f3 && (1/0 > 0))); // No ArithmeticException
        System.out.println("   true || (1/0 > 0) = " + (t3 || (1/0 > 0))); // No ArithmeticException

        // Operator Precedence
        System.out.println("\n3. Operator Precedence:");
        System.out.println("   true || false && false = " + (t3 || f3 && f3)); // && has higher precedence
        System.out.println("   (true || false) && false = " + ((t3 || f3) && f3)); // Parentheses change precedence

        // Combining with Comparison Operators
        int num1 = 8, num2 = 15; // Changed values of integers
        System.out.println("\n4. Combining with Comparison Operators:");
        System.out.println("   (num1 < num2) && (num2 > 0) = " + ((num1 < num2) && (num2 > 0)));
        System.out.println("   (num1 > num2) || (num2 < 20) = " + ((num1 > num2) || (num2 < 20)));

        // Complex Conditions
        boolean cond1 = true, cond2 = false, cond3 = true; // Simplified variable names
        System.out.println("\n5. Complex Conditions:");
        System.out.println("   (cond1 && cond2) || (cond1 && cond3) = " + ((cond1 && cond2) || (cond1 && cond3)));
        System.out.println("   cond1 && (cond2 || cond3) = " + (cond1 && (cond2 || cond3)));
        System.out.println("   !cond1 || (cond2 && !cond3) = " + (!cond1 || (cond2 && !cond3)));

        // Bitwise vs. Logical Operators
        System.out.println("\n6. Bitwise vs. Logical Operators:");
        System.out.println("   true & false = " + (t3 & f3)); // Bitwise AND
        System.out.println("   true | false = " + (t3 | f3)); // Bitwise OR
        System.out.println("   true ^ false = " + (t3 ^ f3)); // Bitwise XOR

        // Short-circuit vs. Non-short-circuit
        int count = 0;
        boolean res1 = (f3 && (++count > 0)); // count is not incremented
        boolean res2 = (f3 & (++count > 0));  // count is incremented
        System.out.println("\n7. Short-circuit vs. Non-short-circuit:");
        System.out.println("   Short-circuit AND result: " + res1 + ", count = " + count);
        System.out.println("   Non-short-circuit AND result: " + res2 + ", count = " + count);

        // Logical Operators with Non-boolean Operands
        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')));

        // Logical Operators in Control Structures
        System.out.println("\n9. Logical Operators in Control Structures:");
        if (t3 && !f3) {
            System.out.println("   This will be printed.");
        }

        // While loop with logical operators
        int loopCounter = 0;
        while (loopCounter < 3 && t3) {
            System.out.println("   loopCounter = " + loopCounter);
            loopCounter++;
        }

        // Logical Operators with Method Calls
        System.out.println("\n10. Logical Operators with Method Calls:");
        System.out.println("   isPositive(5) && isEven(4) = " + (isPositive(5) && isEven(4)));
        System.out.println("   isPositive(-2) || isEven(9) = " + (isPositive(-2) || isEven(9)));

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

        // Conditional Assignment with Logical Operators
        int resultValue = t3 ? 1 : 0;
        System.out.println("\n12. Using Logical Operators for Conditional Assignment:");
        System.out.println("   resultValue = " + resultValue);

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

    // Check if a number is positive
    private static boolean isPositive(int num) {
        return num > 0;
    }

    // Check if a number is even
    private static boolean isEven(int num) {
        return num % 2 == 0;
    }
}
```

```java
// CODE 5

class Vehicle {}

public class Automobile extends Vehicle {
    public static void main(String[] args) {
        Vehicle vehicleInstance = new Automobile();  // Create an instance of Automobile as Vehicle type
        boolean isCarInstance = vehicleInstance instanceof Automobile;  // Check if vehicleInstance is an instance of Automobile
        System.out.println("isCarInstance = " + isCarInstance);  // Print the result
    }
}
```

```java
// CODE 6

import java.util.Scanner;

public class CalculatorApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get user's name and greet
        System.out.println("Enter your name: ");
        String userName = scanner.nextLine();
        System.out.println("Hello, " + userName + "!");

        // Calculator functionality
        System.out.println("Enter the first number: ");
        double firstNumber = scanner.nextDouble();
        System.out.println("Enter the second number: ");
        double secondNumber = scanner.nextDouble();
        System.out.println("Enter the operation (+, -, *): ");
        char operationType = scanner.next().charAt(0);

        double calculationResult = 0;
        switch (operationType) {
            case '+':
                calculationResult = firstNumber + secondNumber;
                break;
            case '-':
                calculationResult = firstNumber - secondNumber;
                break;
            case '*':
                calculationResult = firstNumber * secondNumber;
                break;
            default:
                System.out.println("Invalid operation");
                return;
        }
        System.out.println("Result: " + calculationResult);
    }
}
```

```java
// CODE 7

import java.util.Date;
import java.text.SimpleDateFormat;

class DateExample {
    public static void main(String[] args) {
        Date d = new Date();  // Current date and time
        System.out.println("Date: " + d);
        System.out.println("Date: " + d.toString());

        SimpleDateFormat fmt = new SimpleDateFormat("MM-dd-yyyy");  // Date format
        String fDate = fmt.format(d);
        System.out.println("Formatted Date: " + fDate);
    }
}
```

# Group Activity:

Developed a library management system in Java.
The code is attached below.

```java
import java.util.*;

class Book {
    String title;
    String author;
    boolean isIssued;

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.isIssued = false;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public boolean isIssued() {
        return isIssued;
    }

    public void setIssued(boolean issued) {
        this.isIssued = issued;
    }

    @Override
    public String toString() {
        return "Title: " + title + ", Author: " + author + ", Issued: " + isIssued;
    }
}

class Member {
    int id;
    String name;
    String contactNumber;

    public Member(int id, String name, String contactNumber) {
        this.id = id;
        this.name = name;
        this.contactNumber = contactNumber;
    }

    @Override
    public String toString() {
        return "Member ID: " + id + ", Name: " + name + ", Contact: " + contactNumber;
    }
}

class Transaction {
    String bookTitle;
    String memberName;
    String type; // "Issued" or "Returned"
    Date transactionDate;
    Date issueDate; // For the issue date
    Date returnDate; // For the return date

    public Transaction(String bookTitle, String memberName, String type, Date issueDate, Date returnDate) {
        this.bookTitle = bookTitle;
        this.memberName = memberName;
        this.type = type;
        this.transactionDate = new Date();
        this.issueDate = issueDate;
        this.returnDate = returnDate;
    }

    @Override
    public String toString() {
        String issueDateStr = (issueDate != null) ? issueDate.toString() : "N/A";
        String returnDateStr = (returnDate != null) ? returnDate.toString() : "N/A";
        return "Transaction: " + type +
               " | Book: " + bookTitle +
               " | Member: " + memberName +
               " | Date: " + transactionDate +
               " | Issue Date: " + issueDateStr +
               " | Return Date: " + returnDateStr;
    }
}
```

```java
class Library {
    List<Book> books = new ArrayList<>();
    List<Member> members = new ArrayList<>();
    List<Transaction> transactions = new ArrayList<>();

    public void addBook(Book book) {
        books.add(book);
        System.out.println("Book added: " + book);
    }

    public void displayBooks() {
        System.out.println("\n--- List of Books ---");
        for (Book book : books) {
            System.out.println(book);
        }
    }

    public void issueBook(String title, String memberName) {
        for (Book book : books) {
            if (book.getTitle().equalsIgnoreCase(title) && !book.isIssued()) {
                book.setIssued(true);
                transactions.add(new Transaction(title, memberName, "Issued", new Date(), null));
                System.out.println("Book issued: " + title + " to " + memberName);
                return;
            }
        }
        System.out.println("Book not available or already issued.");
    }

    public void returnBook(String title, String memberName) {
        for (Book book : books) {
            if (book.getTitle().equalsIgnoreCase(title) && book.isIssued()) {
                book.setIssued(false);
                transactions.add(new Transaction(title, memberName, "Returned", null, new Date()));
                System.out.println("Book returned: " + title + " by " + memberName);
                return;
            }
        }
        System.out.println("Book not found or not issued.");
    }

    public void addMember(int id, String name, String contactNumber) {
        members.add(new Member(id, name, contactNumber));
        System.out.println("Member added: " + name);
    }

    public void displayMembers() {
        System.out.println("\n--- List of Members ---");
        for (Member member : members) {
            System.out.println(member);
        }
    }

    public void searchByTitle(String title) {
        for (Book book : books) {
            if (book.getTitle().equalsIgnoreCase(title)) {
                System.out.println(book);
                return;
            }
        }
        System.out.println("Book not found.");
    }

    public void searchByAuthor(String author) {
        boolean found = false;
        for (Book book : books) {
            if (book.getAuthor().equalsIgnoreCase(author)) {
                System.out.println(book);
                found = true;
            }
        }
        if (!found) {
            System.out.println("No books found by the author.");
        }
    }

    public void displayTransactions() {
        System.out.println("\n--- Transaction Details ---");
        for (Transaction transaction : transactions) {
            System.out.println(transaction);
        }
    }
}
```

```java
public class LibraryManagementSystem {
    static Scanner scanner = new Scanner(System.in);
    static Library library = new Library();

    public static void main(String[] args) {
        int choice;
        do {
            System.out.println("\n--- Library Management System ---");
            System.out.println("1. Add Book");
            System.out.println("2. Display Books");
            System.out.println("3. Issue Book");
            System.out.println("4. Return Book");
            System.out.println("5. Add Member");
            System.out.println("6. Display Members");
            System.out.println("7. Search by Title");
            System.out.println("8. Search by Author");
            System.out.println("9. Display Transactions");
            System.out.println("10. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter book title: ");
                    String title = scanner.nextLine();
                    System.out.print("Enter book author: ");
                    String author = scanner.nextLine();
                    library.addBook(new Book(title, author));
                    break;
                case 2:
                    library.displayBooks();
                    break;
                case 3:
                    System.out.print("Enter book title to issue: ");
                    title = scanner.nextLine();
                    System.out.print("Enter member name: ");
                    String memberName = scanner.nextLine();
                    library.issueBook(title, memberName);
                    break;
                case 4:
                    System.out.print("Enter book title to return: ");
                    title = scanner.nextLine();
                    System.out.print("Enter member name: ");
                    memberName = scanner.nextLine();
                    library.returnBook(title, memberName);
                    break;
                case 5:
                    System.out.print("Enter member ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Enter member name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter contact number: ");
                    String contactNumber = scanner.nextLine();
                    library.addMember(id, name, contactNumber);
                    break;
                case 6:
                    library.displayMembers();
                    break;
                case 7:
                    System.out.print("Enter book title to search: ");
                    title = scanner.nextLine();
                    library.searchByTitle(title);
                    break;
                case 8:
                    System.out.print("Enter author name to search: ");
                    author = scanner.nextLine();
                    library.searchByAuthor(author);
                    break;
                case 9:
                    library.displayTransactions();
                    break;
                case 10:
                    System.out.println("Exiting system...");
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        } while (choice != 10);

        scanner.close();
    }
}
```

# DAY 4 (19 September)

```java
// CODE 8
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;

public class JavaControlStatements {

    public static void main(String[] args) {
        // 1. Conditional Statements
        conditionalStatements();

        // 2. Looping Statements
        loopingStatements();

        // 3. Jump Statements
        jumpStatements();

        // 4. Exception Handling
        exceptionHandling();

        // 5. Assertions
        assertions();
    }

    // 1. Conditional Statements
    private static void conditionalStatements() {
        System.out.println("\n--- Conditional Statements ---");

        // 1.1 if statement
        int age = 18;
        if (age >= 18) {
            System.out.println("You are eligible to vote.");
        }

        // 1.2 if-else statement
        int score = 75;
        if (score >= 60) {
            System.out.println("You passed!");
        } else {
            System.out.println("You failed.");
        }

        // 1.3 if-else-if ladder
        int grade = 85;
        if (grade >= 90) {
            System.out.println("Grade: A");
        } else if (grade >= 80) {
            System.out.println("Grade: B");
        } else if (grade >= 70) {
            System.out.println("Grade: C");
        } else if (grade >= 60) {
            System.out.println("Grade: D");
        } else {
            System.out.println("Grade: F");
        }

        // 1.4 Nested if statements
        boolean hasLicense = true;
        if (age >= 18) {
            if (hasLicense) {
                System.out.println("You can drive.");
            } else {
                System.out.println("You need to get a license first.");
            }
        } else {
            System.out.println("You are too young to drive.");
        }

        // 1.5 switch statement
        int dayOfWeek = 3;
        switch (dayOfWeek) {
            case 1:
                System.out.println("Day: Monday");
                break;
            case 2:
                System.out.println("Day: Tuesday");
                break;
            case 3:
                System.out.println("Day: Wednesday");
                break;
            case 4:
                System.out.println("Day: Thursday");
                break;
            case 5:
                System.out.println("Day: Friday");
                break;
            case 6:
            case 7:
                System.out.println("Day: Weekend");
                break;
            default:
                System.out.println("Invalid day");
        }
    }
```

```java
// 2. Looping Statements
    private static void loopingStatements() {
        System.out.println("\n--- Looping Statements ---");

        // 2.1 for loop
        System.out.println("For loop:");
        for (int i = 1; i <= 5; i++) {
            System.out.println("Iteration: " + i);
        }

        // 2.2 Enhanced for loop (for-each)
        System.out.println("\nEnhanced for loop:");
        int[] numbers = {1, 2, 3, 4, 5};
        for (int num : numbers) {
            System.out.println("Number: " + num);
        }

        // 2.3 while loop
        System.out.println("\nWhile loop:");
        int count = 0;
        while (count < 5) {
            System.out.println("Count: " + count);
            count++;
        }

        // 2.4 do-while loop
        System.out.println("\nDo-while loop:");
        int num = 1;
        do {
            System.out.println("Number: " + num);
            num++;
        } while (num <= 5);
    }

    // 3. Jump Statements
    private static void jumpStatements() {
        System.out.println("\n--- Jump Statements ---");

        // 3.1 break statement
        System.out.println("Break statement:");
        for (int i = 1; i <= 10; i++) {
            if (i == 6) {
                break;
            }
            System.out.println("Iteration: " + i);
        }

        // 3.2 continue statement
        System.out.println("\nContinue statement:");
        for (int i = 1; i <= 5; i++) {
            if (i == 3) {
                continue;
            }
            System.out.println("Iteration: " + i);
        }

        // 3.3 return statement
        System.out.println("\nReturn statement:");
        System.out.println("Sum: " + sum(5, 3));
    }

    private static int sum(int a, int b) {
        return a + b;
    }
```

```java
// 4. Exception Handling
    private static void exceptionHandling() {
        System.out.println("\n--- Exception Handling ---");

        // 4.1 try-catch
        System.out.println("Try-catch:");
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        }

        // 4.2 try-catch-finally
        System.out.println("\nTry-catch-finally:");
        try {
            int[] numbers = {1, 2, 3};
            System.out.println(numbers[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            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("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);
    }
}
```

```java
// CODE 9
import java.util.Scanner;
import java.util.Random;

public class ATMSimulator {
    private static final int PIN = 1234; // Simulated correct PIN
    private static final int MAX_ATTEMPTS = 3;
    private static double balance = 1000.00; // Initial balance

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();

        System.out.println("Welcome to the ATM Simulator");

        // PIN verification using do-while loop
        int attempts = 0;
        boolean pinVerified = false;
        do {
            System.out.print("Please enter your PIN: ");
            int enteredPin = scanner.nextInt();

            if (enteredPin == PIN) {
                pinVerified = true;
                System.out.println("PIN accepted. Access granted.");
            } else {
                attempts++;
                System.out.println("Incorrect PIN. Attempts remaining: " + (MAX_ATTEMPTS - attempts));
            }
        } while (!pinVerified && attempts < MAX_ATTEMPTS);

        if (!pinVerified) {
            System.out.println("Too many incorrect attempts. Your card has been blocked.");
            return;
        }

        // Main ATM menu using do-while loop
        int choice;
        do {
            System.out.println("\nATM Menu:");
            System.out.println("1. Check Balance");
            System.out.println("2. Withdraw Money");
            System.out.println("3. Deposit Money");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.printf("Your current balance is: $%.2f\n", balance);
                    break;
                case 2:
                    System.out.print("Enter amount to withdraw: $");
                    double withdrawAmount = scanner.nextDouble();
                    if (withdrawAmount > balance) {
                        System.out.println("Insufficient funds.");
                    } else {
                        balance -= withdrawAmount;
                        System.out.printf("$%.2f withdrawn. New balance: $%.2f\n", withdrawAmount, balance);
                    }
                    break;
                case 3:
                    System.out.print("Enter amount to deposit: $");
                    double depositAmount = scanner.nextDouble();
                    balance += depositAmount;
                    System.out.printf("$%.2f deposited. New balance: $%.2f\n", depositAmount, balance);
                    break;
                case 4:
                    System.out.println("Thank you for using the ATM. Goodbye!");
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }

            // Simulate processing time
            try {
                Thread.sleep(random.nextInt(1000) + 500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

        } while (choice != 4);

        scanner.close();
    }
}
```

```java
// CODE 10
// Java OOP Concepts for Beginners - Comprehensive Guide

// This program demonstrates Object-Oriented Programming (OOP) concepts in Java
// in an order suitable for beginners to understand. Each concept is explained
// with comments and multiple examples.

// 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.

class Car {
    // Fields (attributes)
    String brand;
    String model;
    int year;

    // Constructor: special method to initialize objects
    Car(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    // Method (behavior)
    void displayInfo() {
        System.out.println("Car: " + year + " " + brand + " " + model);
    }
}

// 2. Encapsulation
// Encapsulation is the bundling of data and the methods that operate on that data within a single unit (class).
// It restricts direct access to some of an object's components, which is a means of preventing accidental
// interference and misuse of the methods and data.

class BankAccount {
    // Private fields - can only be accessed within this class
    private String accountNumber;
    private double balance;

    // Constructor
    public BankAccount(String accountNumber, double initialBalance) {
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }

    // Public methods to access and modify the private fields
    public String getAccountNumber() {
        return accountNumber;
    }

    public double getBalance() {
        return balance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
        }
    }
}
```

```java
// 3. Inheritance
// Inheritance is a mechanism where a new class is derived from an existing class.
// The new class (subclass) inherits fields and methods from the existing class (superclass).

// Superclass
class Animal {
    String name;

    Animal(String name) {
        this.name = name;
    }

    void eat() {
        System.out.println(name + " is eating.");
    }
}

// Subclass
class Dog extends Animal {
    Dog(String name) {
        super(name); // Call the superclass constructor
    }

    void bark() {
        System.out.println(name + " is barking.");
    }
}

// 4. Polymorphism
// Polymorphism allows objects of different classes to be treated as objects of a common superclass.
// It can be achieved through method overriding and method overloading.

// Method Overriding: providing a specific implementation of a method in the subclass
// that is already defined in the superclass.
class Cat extends Animal {
    Cat(String name) {
        super(name);
    }

    // Override the eat() method
    @Override
    void eat() {
        System.out.println(name + " is eating fish.");
    }

    void meow() {
        System.out.println(name + " is meowing.");
    }
}

// 5. Abstraction
// Abstraction is the process of hiding the implementation details and showing only the functionality to the user.
// It can be achieved through abstract classes and interfaces.

// Abstract class
abstract class Shape {
    abstract double calculateArea();

    void display() {
        System.out.println("This is a shape.");
    }
}

class Circle extends Shape {
    double radius;

    Circle(double radius) {
        this.radius = radius;
    }

    @Override
    double calculateArea() {
        return Math.PI * radius * radius;
    }
}
```

```java
// 6. Interfaces
// An interface is a completely abstract class that contains only abstract methods.
// It can be used to achieve multiple inheritance in Java.

interface Drawable {
    void draw();
}

class Rectangle extends Shape implements Drawable {
    double length;
    double width;

    Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    @Override
    double calculateArea() {
        return length * width;
    }

    @Override
    public void draw() {
        System.out.println("Drawing a rectangle");
    }
}

// Main class to demonstrate all concepts
public class JavaOOPConceptsForBeginners {
    public static void main(String[] args) {
        // 1. Classes and Objects
        System.out.println("1. Classes and Objects:");
        Car myCar = new Car("Toyota", "Corolla", 2022);
        myCar.displayInfo();

        // 2. Encapsulation
        System.out.println("\n2. Encapsulation:");
        BankAccount account = new BankAccount("123456", 1000);
        account.deposit(500);
        account.withdraw(200);
        System.out.println("Account balance: $" + account.getBalance());

        // 3. Inheritance
        System.out.println("\n3. Inheritance:");
        Dog myDog = new Dog("Buddy");
        myDog.eat();
        myDog.bark();

        // 4. Polymorphism
        System.out.println("\n4. Polymorphism:");
        Animal myAnimal = new Cat("Whiskers");
        myAnimal.eat(); // This will call the overridden method in Cat

        // 5. Abstraction
        System.out.println("\n5. Abstraction:");
        Shape myCircle = new Circle(5);
        System.out.println("Circle area: " + myCircle.calculateArea());

        // 6. Interfaces
        System.out.println("\n6. Interfaces:");
        Rectangle myRectangle = new Rectangle(4, 5);
        myRectangle.draw();
        System.out.println("Rectangle area: " + myRectangle.calculateArea());
    }
}
```

```java
// CODE 11
// Computer System Example with Simplified Variable Names

// Component classes
class Processor {
    private String name;
    private double speedGHz;

    // Constructor for Processor
    public Processor(String name, double speedGHz) {
        this.name = name;
        this.speedGHz = speedGHz;
    }

    // Simulate processing data
    public void runProcess() {
        System.out.println("Processor " + name + " running at " + speedGHz + " GHz");
    }
}

class Memory {
    private String memoryType;
    private int sizeGB;

    // Constructor for Memory
    public Memory(String memoryType, int sizeGB) {
        this.memoryType = memoryType;
        this.sizeGB = sizeGB;
    }

    // Simulate loading data
    public void loadMemory() {
        System.out.println("Loading data into " + sizeGB + "GB " + memoryType + " memory");
    }
}

class Disk {
    private String diskType;
    private int storageGB;

    // Constructor for Disk
    public Disk(String diskType, int storageGB) {
        this.diskType = diskType;
        this.storageGB = storageGB;
    }

    // Simulate storing data
    public void saveData() {
        System.out.println("Saving data on " + storageGB + "GB " + diskType + " storage");
    }
}

class GraphicsCard {
    private String graphicsModel;
    private int memorySizeGB;

    // Constructor for GraphicsCard
    public GraphicsCard(String graphicsModel, int memorySizeGB) {
        this.graphicsModel = graphicsModel;
        this.memorySizeGB = memorySizeGB;
    }
```

```java
public void displayGraphics() {
        System.out.println("Rendering with " + graphicsModel + " using " + memorySizeGB + "GB memory");
    }
}

// Main class using components
class PC {
    private Processor processor;
    private Memory memory;
    private Disk disk;
    private GraphicsCard graphicsCard;

    // Constructor for PC
    public PC(Processor processor, Memory memory, Disk disk, GraphicsCard graphicsCard) {
        this.processor = processor;
        this.memory = memory;
        this.disk = disk;
        this.graphicsCard = graphicsCard;
    }

    // Simulate booting up the computer
    public void startComputer() {
        System.out.println("Starting the PC...");
        processor.runProcess();
        memory.loadMemory();
        disk.saveData();
        graphicsCard.displayGraphics();
        System.out.println("PC is ready for use!");
    }

    // Simulate shutting down the computer
    public void powerOff() {
        System.out.println("Powering down the PC...");
        disk.saveData();  // Ensure data is saved
        System.out.println("PC has been powered off.");
    }

    // Getter methods for components
    public Processor getProcessor() { return processor; }
    public Memory getMemory() { return memory; }
    public Disk getDisk() { return disk; }
    public GraphicsCard getGraphicsCard() { return graphicsCard; }
}

// Example usage of the new PC system
public class PCExample {
    public static void main(String[] args) {
        Processor processor = new Processor("AMD Ryzen 5", 4.2); // Changed model and speed
        Memory memory = new Memory("DDR5", 32); // Changed type and capacity
        Disk disk = new Disk("NVMe", 1024); // Changed type and capacity
        GraphicsCard graphicsCard = new GraphicsCard("AMD Radeon RX 6800", 12); // Changed model and memory

        PC userPC = new PC(processor, memory, disk, graphicsCard);

        userPC.startComputer();
        System.out.println("\nPerforming additional tasks...");
        userPC.getProcessor().runProcess();
        userPC.getGraphicsCard().displayGraphics();
        System.out.println();
        userPC.powerOff();
    }
}
```

```java
// CODE 12
// Examples of Abstract Class and Interface Usage

// Example 1: Shape hierarchy
abstract class Figure {
    protected String shade; // Color of the shape

    public Figure(String shade) {
        this.shade = shade;
    }

    // Abstract method to calculate area
    public abstract double getArea();

    // Concrete method to display color
    public void showColor() {
        System.out.println("Color: " + shade);
    }
}

class RoundShape extends Figure {
    private double rad; // Radius of the circle

    public RoundShape(String shade, double rad) {
        super(shade);
        this.rad = rad;
    }

    @Override
    public double getArea() {
        return Math.PI * rad * rad;
    }
}

class BoxShape extends Figure {
    private double length;
    private double width;

    public BoxShape(String shade, double length, double width) {
        super(shade);
        this.length = length;
        this.width = width;
    }

    @Override
    public double getArea() {
        return length * width;
    }
}

// Example 2: Vehicle interface and implementations
interface Transport {
    void ignite();
    void halt();
    int fuelStatus();
}

abstract class MotorVehicle implements Transport {
    protected int fuelLevel;

    public MotorVehicle(int startFuel) {
        this.fuelLevel = startFuel;
    }

    @Override
    public int fuelStatus() {
        return fuelLevel;
    }

    // Abstract method for refueling
    public abstract void refill(int amount);
}
```

```java
class Sedan extends MotorVehicle {
    public Sedan(int startFuel) {
        super(startFuel);
    }

    @Override
    public void ignite() {
        System.out.println("Sedan engine started");
    }

    @Override
    public void halt() {
        System.out.println("Sedan engine stopped");
    }

    @Override
    public void refill(int amount) {
        fuelLevel += amount;
        System.out.println("Sedan refueled. Fuel level: " + fuelLevel);
    }
}

class PedalBike implements Transport {
    @Override
    public void ignite() {
        System.out.println("Bicycle in motion");
    }

    @Override
    public void halt() {
        System.out.println("Bicycle stopped");
    }

    @Override
    public int fuelStatus() {
        return 100; // No fuel needed for a bicycle
    }
}

// Example 3: Payment processing
interface PayProcessor {
    boolean process(double amount);
    void returnPayment(double amount);
}

abstract class OnlinePayProcessor implements PayProcessor {
    protected String merchantID;

    public OnlinePayProcessor(String merchantID) {
        this.merchantID = merchantID;
    }

    // Abstract method to verify merchant
    public abstract boolean validateMerchant();
}

class CardProcessor extends OnlinePayProcessor {
    public CardProcessor(String merchantID) {
        super(merchantID);
    }

    @Override
    public boolean process(double amount) {
        if (validateMerchant()) {
            System.out.println("Processing card payment of $" + amount);
            return true;
        }
        return false;
    }
```

```java
    @Override
    public void returnPayment(double amount) {
        System.out.println("Refunding card payment of $" + amount);
    }

    @Override
    public boolean validateMerchant() {
        return merchantID.startsWith("CRD");
    }
}

class PayPalProcessor extends OnlinePayProcessor {
    public PayPalProcessor(String merchantID) {
        super(merchantID);
    }

    @Override
    public boolean process(double amount) {
        if (validateMerchant()) {
            System.out.println("Processing PayPal payment of $" + amount);
            return true;
        }
        return false;
    }

    @Override
    public void returnPayment(double amount) {
        System.out.println("Refunding PayPal payment of $" + amount);
    }

    @Override
    public boolean validateMerchant() {
        return merchantID.startsWith("PPL");
    }
}

public class AbstractAndInterfaceExample {
    public static void main(String[] args) {
        // Example 1: Shape hierarchy
        System.out.println("Example 1: Shape Hierarchy");
        Figure round = new RoundShape("Green", 7); // Changed values
        Figure box = new BoxShape("Yellow", 5, 8); // Changed values

        System.out.println("Circle:");
        round.showColor();
        System.out.println("Area: " + round.getArea());

        System.out.println("\nRectangle:");
        box.showColor();
        System.out.println("Area: " + box.getArea());

        // Example 2: Vehicle interface and implementations
        System.out.println("\nExample 2: Vehicle Interface");
        Transport sedan = new Sedan(40); // Changed fuel level
        Transport bike = new PedalBike();

        sedan.ignite();
        System.out.println("Sedan fuel level: " + sedan.fuelStatus());
        ((Sedan) sedan).refill(15); // Changed refuel amount
        sedan.halt();

        bike.ignite();
        System.out.println("Bicycle 'fuel' level: " + bike.fuelStatus());
        bike.halt();

        // Example 3: Payment processing
        System.out.println("\nExample 3: Payment Processing");
        PayProcessor cardPay = new CardProcessor("CRD987");
        PayProcessor payPalPay = new PayPalProcessor("PPL123");

        cardPay.process(150.75); // Changed payment amount
        cardPay.returnPayment(35.50); // Changed refund amount

        payPalPay.process(95.30); // Changed payment amount
        payPalPay.returnPayment(95.30);
    }
}
```

```java
// CODE 13
// Java Parameterized Constructors: Detailed Examples and Use Cases

class Book {
    private String title;
    private String author;
    private int pageCount;
    private String isbn;
    private boolean isHardcover;

    // Parameterized constructor with all attributes
    public Book(String title, String author, int pageCount, String isbn, boolean isHardcover) {
        this.title = title;
        this.author = author;
        this.pageCount = pageCount;
        this.isbn = isbn;
        this.isHardcover = isHardcover;
    }

    // Parameterized constructor with essential attributes
    public Book(String title, String author, String isbn) {
        this(title, author, 0, isbn, false); // Calls the full constructor with default values
    }

    // Parameterized constructor for a book without known ISBN
    public Book(String title, String author, int pageCount) {
        this(title, author, pageCount, "Unknown", false);
    }

    // Copy constructor
    public Book(Book otherBook) {
        this(otherBook.title, otherBook.author, otherBook.pageCount, otherBook.isbn, otherBook.isHardcover);
    }

    @Override
    public String toString() {
        return "Book{" +
               "title='" + title + '\'' +
               ", author='" + author + '\'' +
               ", pageCount=" + pageCount +
               ", isbn='" + isbn + '\'' +
               ", isHardcover=" + isHardcover +
               '}';
    }
}
```

```java
class BankAccount {
    private String accountNumber;
    private String accountHolder;
    private double balance;
    private String accountType;

    // Parameterized constructor for creating a new account
    public BankAccount(String accountHolder, String accountType, double initialDeposit) {
        this.accountNumber = generateAccountNumber();
        this.accountHolder = accountHolder;
        this.accountType = accountType;
        this.balance = initialDeposit;
    }

    // Parameterized constructor for loading an existing account
    public BankAccount(String accountNumber, String accountHolder, double balance, String accountType) {
        this.accountNumber = accountNumber;
        this.accountHolder = accountHolder;
        this.balance = balance;
        this.accountType = accountType;
    }

    private String generateAccountNumber() {
        // Simulating account number generation
        return "ACC" + Math.round(Math.random() * 1000000);
    }

    @Override
    public String toString() {
        return "BankAccount{" +
               "accountNumber='" + accountNumber + '\'' +
               ", accountHolder='" + accountHolder + '\'' +
               ", balance=" + balance +
               ", accountType='" + accountType + '\'' +
               '}';
    }
}

public class ParameterizedConstructorExamples {
    public static void main(String[] args) {
        // Book examples
        Book book1 = new Book("The Great Gatsby", "F. Scott Fitzgerald", 180, "9780743273565", true);
        Book book2 = new Book("To Kill a Mockingbird", "Harper Lee", "9780446310789");
        Book book3 = new Book("1984", "George Orwell", 328);
        Book book4 = new Book(book1); // Using copy constructor

        System.out.println("Book Examples:");
        System.out.println(book1);
        System.out.println(book2);
        System.out.println(book3);
        System.out.println(book4);

        // BankAccount examples
        BankAccount account1 = new BankAccount("John Doe", "Savings", 1000.0);
        BankAccount account2 = new BankAccount("ACC123456", "Jane Smith", 5000.0, "Checking");

        System.out.println("\nBank Account Examples:");
        System.out.println(account1);
        System.out.println(account2);
    }
}
```

```java
// CODE 14
// File: AccessModifiersDemo.java

// The public class AccessModifiersDemo can be accessed from any other class.
public class AccessModifiersDemo {

    // Public member variable
    // Can be accessed from any other class
    public int publicVar = 10;

    // Protected member variable
    // Can be accessed within the same package and by subclasses (even if in a different package)
    protected int protectedVar = 20;

    // Default (package-private) member variable
    // Can be accessed only within the same package
    int defaultVar = 30;

    // Private member variable
    // Can be accessed only within this class
    private int privateVar = 40;

    // Public method
    // Can be called from any other class
    public void publicMethod() {
        System.out.println("This is a public method");
        // Public methods can access all members of the class
        System.out.println("Accessing all variables:");
        System.out.println("publicVar: " + publicVar);
        System.out.println("protectedVar: " + protectedVar);
        System.out.println("defaultVar: " + defaultVar);
        System.out.println("privateVar: " + privateVar);
    }

    // Protected method
    // Can be called within the same package and by subclasses
    protected void protectedMethod() {
        System.out.println("This is a protected method");
    }

    // Default (package-private) method
    // Can be called only within the same package
    void defaultMethod() {
        System.out.println("This is a default method");
    }

    // Private method
    // Can be called only within this class
    private void privateMethod() {
        System.out.println("This is a private method");
    }

    // Inner class to demonstrate access modifiers within the same class
    public class InnerClass {
        public void accessMembers() {
            // Inner classes can access all members of the enclosing class,
            // including private members
            System.out.println("InnerClass accessing privateVar: " + privateVar);
            privateMethod();
        }
    }
}

// This class is in the same file to demonstrate default access
class SameFileClass {
    public void accessMembers() {
        AccessModifiersDemo demo = new AccessModifiersDemo();
        // Can access public, protected, and default members
        System.out.println("SameFileClass accessing publicVar: " + demo.publicVar);
        System.out.println("SameFileClass accessing protectedVar: " + demo.protectedVar);
        System.out.println("SameFileClass accessing defaultVar: " + demo.defaultVar);
        // Cannot access private members
        // System.out.println(demo.privateVar); // This would cause a compilation error

        demo.publicMethod();
        demo.protectedMethod();
        demo.defaultMethod();
        // demo.privateMethod(); // This would cause a compilation error
    }
}
```

```java
// File: SubClass.java (Assume this is in the same package)

// Subclass in the same package
class SubClass extends AccessModifiersDemo {
    public void accessMembers() {
        // Can access public, protected, and default members of the superclass
        System.out.println("SubClass accessing publicVar: " + publicVar);
        System.out.println("SubClass accessing protectedVar: " + protectedVar);
        System.out.println("SubClass accessing defaultVar: " + defaultVar);
        // Cannot access private members of the superclass
        // System.out.println(privateVar); // This would cause a compilation error

        publicMethod();
        protectedMethod();
        defaultMethod();
        // privateMethod(); // This would cause a compilation error
    }
}

// File: OtherPackageClass.java (Assume this is in a different package)

// Uncomment the following line when actually placing this class in another package
// package com.example.otherpackage;

// import com.example.AccessModifiersDemo;

// Class in a different package
public class OtherPackageClass {
    public void accessMembers() {
        AccessModifiersDemo demo = new AccessModifiersDemo();
        // Can only access public members
        System.out.println("OtherPackageClass accessing publicVar: " + demo.publicVar);
        // Cannot access protected, default, or private members
        // System.out.println(demo.protectedVar); // Compilation error
        // System.out.println(demo.defaultVar);   // Compilation error
        // System.out.println(demo.privateVar);   // Compilation error

        demo.publicMethod();
        // demo.protectedMethod(); // Compilation error
        // demo.defaultMethod();   // Compilation error
        // demo.privateMethod();   // Compilation error
    }
}

// File: OtherPackageSubClass.java (Assume this is in a different package)

// Uncomment the following line when actually placing this class in another package
// package com.example.otherpackage;

// import com.example.AccessModifiersDemo;

// Subclass in a different package
public class OtherPackageSubClass extends AccessModifiersDemo {
    public void accessMembers() {
        // Can access public and protected members of the superclass
        System.out.println("OtherPackageSubClass accessing publicVar: " + publicVar);
        System.out.println("OtherPackageSubClass accessing protectedVar: " + protectedVar);
        // Cannot access default or private members of the superclass
        // System.out.println(defaultVar);  // Compilation error
        // System.out.println(privateVar);  // Compilation error

        publicMethod();
        protectedMethod();
        // defaultMethod();  // Compilation error
        // privateMethod();  // Compilation error
    }
}
```

```java
// Code 15
import java.io.*;
import java.nio.file.*;
import java.util.Scanner;
import java.util.stream.Stream;

public class JavaIOExamples {

    public static void main(String[] args) {
        String fileName = "example.txt";
        String content = "Hello, Java I/O!";

        // Writing to a file
        writeFile(fileName, content);

        // Reading from a file using various methods
        System.out.println("1. Using FileReader and BufferedReader:");
        readWithBufferedReader(fileName);

        System.out.println("\n2. Using FileInputStream:");
        readWithFileInputStream(fileName);

        System.out.println("\n3. Using Scanner:");
        readWithScanner(fileName);

        System.out.println("\n4. Using Files.readAllLines (Java NIO):");
        readWithFilesReadAllLines(fileName);

        System.out.println("\n5. Using Files.lines (Java NIO) with Stream API:");
        readWithFilesLines(fileName);

        // Demonstrating file operations
        fileOperations(fileName);
    }

    public static void writeFile(String fileName, String content) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {
            writer.write(content);
            System.out.println("Content written to file successfully.");
        } catch (IOException e) {
            System.err.println("Error writing to file: " + e.getMessage());
        }
    }

    public static void readWithBufferedReader(String fileName) {
        try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
        }
    }

    public static void readWithFileInputStream(String fileName) {
        try (FileInputStream fis = new FileInputStream(fileName)) {
            int content;
            while ((content = fis.read()) != -1) {
                System.out.print((char) content);
            }
            System.out.println();
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
        }
    }
```

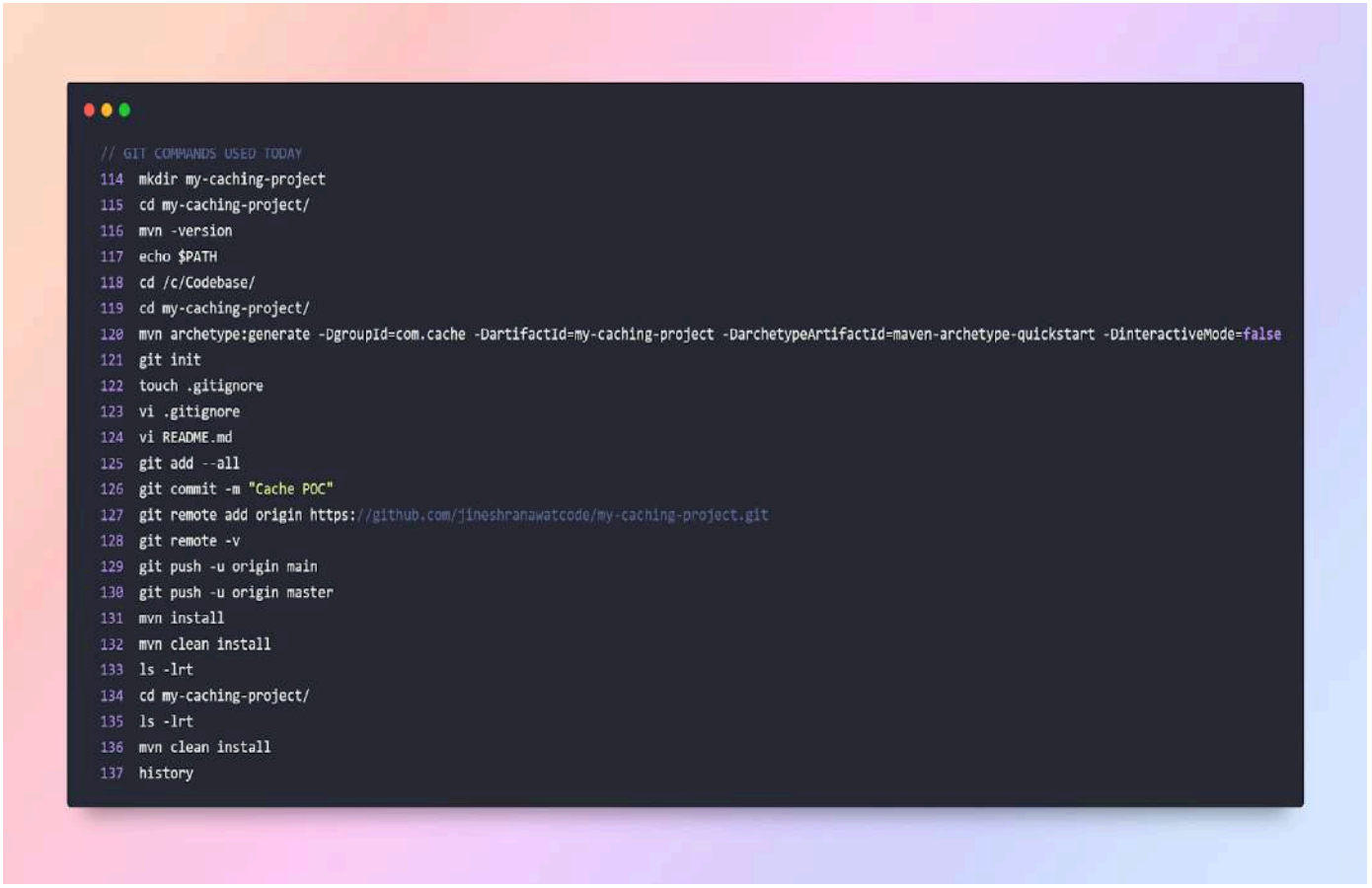
```java
public static void readWithScanner(String fileName) {
        try (Scanner scanner = new Scanner(new File(fileName))) {
            while (scanner.hasNextLine()) {
                System.out.println(scanner.nextLine());
            }
        } catch (FileNotFoundException e) {
            System.err.println("File not found: " + e.getMessage());
        }
    }

    public static void readWithFilesReadAllLines(String fileName) {
        try {
            Files.readAllLines(Paths.get(fileName)).forEach(System.out::println);
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
        }
    }

    public static void readWithFilesLines(String fileName) {
        try (Stream<String> stream = Files.lines(Paths.get(fileName))) {
            stream.forEach(System.out::println);
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
        }
    }

    public static void fileOperations(String fileName) {
        File file = new File(fileName);

        System.out.println("\nFile Operations:");
        System.out.println("Exists: " + file.exists());
        System.out.println("Is File: " + file.isFile());
        System.out.println("Is Directory: " + file.isDirectory());
        System.out.println("Can Read: " + file.canRead());
        System.out.println("Can Write: " + file.canWrite());
        System.out.println("Absolute Path: " + file.getAbsolutePath());

        // Rename the file
        File newFile = new File("new_" + fileName);
        if (file.renameTo(newFile)) {
            System.out.println("File renamed successfully.");
        } else {
            System.out.println("Failed to rename the file.");
        }

        // Delete the file
        if (newFile.delete()) {
            System.out.println("File deleted successfully.");
        } else {
            System.out.println("Failed to delete the file.");
        }
    }
}
```

```java
// CODE 16
import java.util.Scanner;

// Custom checked exception
class LowBalanceException extends Exception {
    private double deficit;

    public LowBalanceException(double deficit) {
        super("Not enough balance. You need " + deficit + " more to complete this transaction.");
        this.deficit = deficit;
    }

    public double getDeficit() {
        return deficit;
    }
}

// Custom unchecked exception
class InvalidUserException extends RuntimeException {
    private String userId;

    public InvalidUserException(String userId) {
        super("Invalid user ID: " + userId);
        this.userId = userId;
    }

    public String getUserId() {
        return userId;
    }
}

// Main class demonstrating exception handling
public class UserAccountExceptionDemo {

    public static void main(String[] args) {
        UserAccount userAccount = new UserAccount("98765", 1500.0);
        Scanner input = new Scanner(System.in);

        while (true) {
            try {
                // Demonstrate handling multiple types of exceptions
                System.out.print("Please enter your user ID: ");
                String userId = input.nextLine();
                System.out.print("Enter amount to withdraw: ");
                double withdrawAmount = Double.parseDouble(input.nextLine());

                // This may throw InvalidUserException or LowBalanceException
                userAccount.withdrawFunds(userId, withdrawAmount);

                System.out.println("Withdrawal completed. Your new balance is: $" + userAccount.getBalance());
                break;
            } catch (LowBalanceException e) {
                // Handle the custom checked exception
                System.out.println("Error: " + e.getMessage());
                System.out.println("You need $" + e.getDeficit() + " more. Try again? (y/n)");
                if (!input.nextLine().equalsIgnoreCase("y")) {
                    break;
                }
            } catch (InvalidUserException e) {
                // Handle the custom unchecked exception
                System.out.println("Error: " + e.getMessage());
                System.out.println("Invalid user ID. Please try again.");
            } catch (NumberFormatException e) {
                // Handle invalid input for withdrawal amount
                System.out.println("Error: Please enter a valid numeric amount.");
            } catch (Exception e) {
                // Generic exception handler for any unexpected exceptions
                System.out.println("An unknown error occurred: " + e.getMessage());
                e.printStackTrace();
                break;
            } finally {
                // This block always executes
                System.out.println("Attempt to process transaction completed.");
            }
        }

        input.close();
    }
}
```

```java
// User account class with methods that may throw exceptions
class UserAccount {
    private String userId;
    private double balance;

    public UserAccount(String userId, double initialBalance) {
        this.userId = userId;
        this.balance = initialBalance;
    }

    public double getBalance() {
        return balance;
    }

    // This method demonstrates throwing both custom checked and unchecked exceptions
    public void withdrawFunds(String userId, double amount) throws LowBalanceException {
        // Validate user ID (may throw unchecked exception)
        if (!this.userId.equals(userId)) {
            throw new InvalidUserException(userId);
        }

        // Check for sufficient funds (may throw checked exception)
        if (amount > balance) {
            double shortage = amount - balance;
            throw new LowBalanceException(shortage);
        }

        // Perform withdrawal if all checks pass
        balance -= amount;
    }
}
```

## DAY 5 (20 September)

```java
// CODE 17
import java.util.Scanner;

class PizzaShop {
    // Enum definition for pizza sizes
    public enum PizzaSize {
        SMALL(8.0, 10),
        MEDIUM(10.0, 12),
        LARGE(12.0, 14),
        EXTRA_LARGE(14.0, 16),
        FAMILY_SIZE(18.0, 20);

        private final double price;
        private final int diameter;

        // Constructor for the enum
        PizzaSize(double price, int diameter) {
            this.price = price;
            this.diameter = diameter;
        }

        // Getter methods
        public double getPrice() {
            return price;
        }

        public int getDiameter() {
            return diameter;
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Welcome to our Pizza Shop!");
        System.out.println("Available sizes:");

        // Display all available pizza sizes
        for (PizzaSize size : PizzaSize.values()) {
            System.out.printf("%s (%d inches) - $%.2f%n",
                    size.name(),
                    size.getDiameter(),
                    size.getPrice());
        }

        System.out.print("Enter your choice (SMALL/MEDIUM/LARGE/EXTRA_LARGE/FAMILY_SIZE): ");
        String userChoice = scanner.nextLine().toUpperCase();

        try {
            PizzaSize selectedSize = PizzaSize.valueOf(userChoice);
            System.out.printf("You've selected a %s pizza (%d inches).%n",
                    selectedSize.name(),
                    selectedSize.getDiameter());
            System.out.printf("Price: $%.2f%n", selectedSize.getPrice());
        } catch (IllegalArgumentException e) {
            System.out.println("Invalid size selected. Please try again.");
        } finally {
            scanner.close();
        }
    }
}
```



```
// GIT COMMANDS USED TODAY
114  mkdir my-caching-project
115  cd my-caching-project/
116  mvn -version
117  echo $PATH
118  cd /c/Codebase/
119  cd my-caching-project/
120  mvn archetype:generate -DgroupId=com.cache -DartifactId=my-caching-project -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
121  git init
122  touch .gitignore
123  vi .gitignore
124  vi README.md
125  git add --all
126  git commit -m "Cache POC"
127  git remote add origin https://github.com/jineshranawatcode/my-caching-project.git
128  git remote -v
129  git push -u origin main
130  git push -u origin master
131  mvn install
132  mvn clean install
133  ls -lrt
134  cd my-caching-project/
135  ls -lrt
136  mvn clean install
137  history
```

```java
// Code 18
package com.cache;
import java.io.*;
import java.util.*;
public class MovieRecommendationCache {
    /**
     * Cache for storing recent movie recommendations.
     * This cache uses a LinkedHashMap with the following characteristics:
     *
     * @param initialCapacity 100 - The initial capacity of the map.
     * @param loadFactor 0.75f - The load factor used to determine when to resize the map.
     * @param accessOrder true - Enables access-order iteration (least recently used order).
     *
     * How it works:
     * 1. The cache stores movie recommendations as key-value pairs (String, String).
     * 2. It has a fixed capacity of 100 entries.
     * 3. When the cache reaches capacity and a new entry is added, the least recently accessed
     *    entry is automatically removed.
     * 4. The load factor of 0.75 means the map will resize when it's 75% full, improving performance.
     * 5. Access-order iteration ensures that entries are ordered based on their access time,
     *    with the least recently used entry at the beginning and the most recently used at the end.
     */
    //L1 Cache Implementation
    private static final int MAX_ENTRIES = 100;
    private final Map<String, String> recentMoviewCache =
        new LinkedHashMap<String, String>(100, 0.75f, true){
            protected boolean removeEldestEntry(Map.Entry<String, String> eldest){
                return size() > MAX_ENTRIES;
            }
        };
    //L2 Cache Implementation
    private static final int MAX_ENTRIES_L2 = 1000;
    private final Map<String, String> popularMoviewCache =
        new LinkedHashMap<String, String>(MAX_ENTRIES_L2, 0.75f, true){
            protected boolean removeEldestEntry(Map.Entry<String, String> eldest){
                return size() > MAX_ENTRIES_L2;
            }
        };
}
```

```java
// Code 19
public static void main(String[] args) {

      Map<String, String> recentMoviewCache =
        new LinkedHashMap<String, String>(100, 0.75f, true);

    //L2 Cache Implementation

      Map<String, String> popularMoviewCache =
        new LinkedHashMap<String, String>(1000, 0.75f, true);

        String[] genres= {"Action", "Comedy", "Drama", "Horror", "Romance", "Sci-Fi", "Thriller"};
        //Write code to generate 1000 movies with random genres and push in popularMoviewCache
        for(int i=0; i<2000; i++){
            String movie = "Movie" + i;
            String genre = genres[new Random().nextInt(genres.length)];
            popularMoviewCache.put(movie, genre);
        }
        //Write code to generate 100 recentlye viewed movies and push in recentMoviewCache
        for(int i=0; i<100; i++){
            String movie = "Movie" + i;
            recentMoviewCache.put(movie, "Recently Viewed");
        }
        //Write code to print it with time it takes to fetch the movie form map
        long startTime = System.nanoTime();
        String movie = popularMoviewCache.get("Movie100");
        long endTime = System.nanoTime();
        System.out.println("Time taken to fetch the movie: " + (endTime - startTime) + " nanoseconds");
        }
```

```java
// CODE 20
import java.util.*;
import java.util.concurrent.*;
import java.util.stream.Collectors;

public class CollectionsConcepts {

    public static void main(String[] args) {
        demonstrateList();
        demonstrateSet();
        demonstrateQueue();
        demonstrateMap();
        demonstrateUtilityClasses();
    }

    private static void demonstrateList() {
        System.out.println("\n--- List Interfaces ---");

        // ArrayList: Dynamic array implementation
        List<String> arrayList = new ArrayList<>();
        arrayList.add("Apple");
        arrayList.add("Banana");
        arrayList.add("Cherry");
        System.out.println("ArrayList: " + arrayList);

        // LinkedList: Doubly-linked list implementation
        List<String> linkedList = new LinkedList<>(arrayList);
        linkedList.add(1, "Blueberry");
        System.out.println("LinkedList: " + linkedList);

        // Vector: Thread-safe dynamic array (legacy class)
        Vector<String> vector = new Vector<>(arrayList);
        vector.add("Date");
        System.out.println("Vector: " + vector);

        // Stack: LIFO stack (extends Vector, legacy class)
        Stack<String> stack = new Stack<>();
        stack.push("Elderberry");
        stack.push("Fig");
        System.out.println("Stack: " + stack);
        System.out.println("Popped from stack: " + stack.pop());

        // Operations
        Collections.sort(arrayList);
        System.out.println("Sorted ArrayList: " + arrayList);
        System.out.println("Binary search for 'Cherry': " + Collections.binarySearch(arrayList, "Cherry"));
    }

    private static void demonstrateSet() {
        System.out.println("\n--- Set Interfaces ---");

        // HashSet: Hash table implementation, no guaranteed order
        Set<String> hashSet = new HashSet<>();
        hashSet.add("Red");
        hashSet.add("Green");
        hashSet.add("Blue");
        hashSet.add("Red"); // Duplicate, won't be added
        System.out.println("HashSet: " + hashSet);

        // LinkedHashSet: Hash table and linked list implementation, maintains insertion order
        Set<String> linkedHashSet = new LinkedHashSet<>();
        linkedHashSet.add("Dog");
        linkedHashSet.add("Cat");
        linkedHashSet.add("Bird");
        System.out.println("LinkedHashSet: " + linkedHashSet);
```

```java
// TreeSet: Red-black tree implementation, sorted order
        Set<String> treeSet = new TreeSet<>();
        treeSet.add("Zebra");
        treeSet.add("Elephant");
        treeSet.add("Lion");
        System.out.println("TreeSet: " + treeSet);

        // Operations
        Set<String> set1 = new HashSet<>(Arrays.asList("A", "B", "C"));
        Set<String> set2 = new HashSet<>(Arrays.asList("B", "C", "D"));
        set1.retainAll(set2); // Intersection
        System.out.println("Intersection: " + set1);
    }

    private static void demonstrateQueue() {
        System.out.println("\n--- Queue Interfaces ---");

        // PriorityQueue: Priority heap implementation
        Queue<Integer> priorityQueue = new PriorityQueue<>();
        priorityQueue.offer(5);
        priorityQueue.offer(1);
        priorityQueue.offer(3);
        System.out.println("PriorityQueue: " + priorityQueue);
        System.out.println("Poll from PriorityQueue: " + priorityQueue.poll());

        // LinkedList as Queue: FIFO queue
        Queue<String> queue = new LinkedList<>();
        queue.offer("First");
        queue.offer("Second");
        queue.offer("Third");
        System.out.println("Queue: " + queue);
        System.out.println("Poll from Queue: " + queue.poll());

        // Deque: Double-ended queue
        Deque<String> deque = new ArrayDeque<>();
        deque.addFirst("Front");
        deque.addLast("Back");
        System.out.println("Deque: " + deque);
        System.out.println("Poll first from Deque: " + deque.pollFirst());
        System.out.println("Poll last from Deque: " + deque.pollLast());

        // BlockingQueue: Thread-safe queue with blocking operations
        BlockingQueue<String> blockingQueue = new LinkedBlockingQueue<>();
        blockingQueue.offer("Task 1");
        blockingQueue.offer("Task 2");
        System.out.println("BlockingQueue: " + blockingQueue);
    }

    private static void demonstrateMap() {
        System.out.println("\n--- Map Interfaces ---");

        // HashMap: Hash table implementation
        Map<String, Integer> hashMap = new HashMap<>();
        hashMap.put("One", 1);
        hashMap.put("Two", 2);
        hashMap.put("Three", 3);
        System.out.println("HashMap: " + hashMap);

        // LinkedHashMap: Hash table and linked list implementation, maintains insertion order
        Map<String, String> linkedHashMap = new LinkedHashMap<>();
        linkedHashMap.put("USA", "Washington D.C.");
        linkedHashMap.put("UK", "London");
        linkedHashMap.put("Japan", "Tokyo");
        System.out.println("LinkedHashMap: " + linkedHashMap);

        // TreeMap: Red-black tree implementation, sorted order
        Map<String, Double> treeMap = new TreeMap<>();
        treeMap.put("Pi", 3.14159);
        treeMap.put("Phi", 1.61803);
        treeMap.put("e", 2.71828);
        System.out.println("TreeMap: " + treeMap);
```

```java
// Hashtable: Thread-safe hash table (legacy class)
        Hashtable<Integer, String> hashtable = new Hashtable<>();
        hashtable.put(1, "First");
        hashtable.put(2, "Second");
        System.out.println("Hashtable: " + hashtable);

        // ConcurrentHashMap: Thread-safe hash table, better performance than Hashtable
        ConcurrentMap<String, Integer> concurrentMap = new ConcurrentHashMap<>();
        concurrentMap.put("Concurrent", 1);
        concurrentMap.put("Hash", 2);
        concurrentMap.put("Map", 3);
        System.out.println("ConcurrentHashMap: " + concurrentMap);

        // Operations
        System.out.println("Keys in HashMap: " + hashMap.keySet());
        System.out.println("Values in HashMap: " + hashMap.values());
        System.out.println("Entries in HashMap: " + hashMap.entrySet());
    }

    private static void demonstrateUtilityClasses() {
        System.out.println("\n--- Utility Classes ---");

        // Arrays utility class
        int[] numbers = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
        Arrays.sort(numbers);
        System.out.println("Sorted array: " + Arrays.toString(numbers));
        System.out.println("Binary search for 5: " + Arrays.binarySearch(numbers, 5));

        // Collections utility class
        List<String> list = Arrays.asList("apple", "banana", "cherry");
        Collections.reverse(list);
        System.out.println("Reversed list: " + list);
        Collections.shuffle(list);
        System.out.println("Shuffled list: " + list);
        System.out.println("Max element: " + Collections.max(list));
        System.out.println("Min element: " + Collections.min(list));

        // Unmodifiable collections
        List<String> unmodifiableList = Collections.unmodifiableList(new ArrayList<>(list));
        // unmodifiableList.add("date"); // This would throw UnsupportedOperationException

        // Synchronized collections
        List<String> synchronizedList = Collections.synchronizedList(new ArrayList<>(list));
        System.out.println("Synchronized list: " + synchronizedList);

        // Java 8 Streams
        List<Integer> numbers2 = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        List<Integer> evenSquares = numbers2.stream()
                                            .filter(n -> n % 2 == 0)
                                            .map(n -> n * n)
                                            .collect(Collectors.toList());
        System.out.println("Even squares: " + evenSquares);
    }
}
```

```java
// CODE 21
public class ThreadingConcepts {
    // volatile keyword ensures that the variable is always read from main memory
    private volatile boolean flag = false;

    // synchronized method to demonstrate thread synchronization
    public synchronized void synchronizedMethod() {
        System.out.println(Thread.currentThread().getName() + " entered synchronized method");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(Thread.currentThread().getName() + " exiting synchronized method");
    }

    // Runnable interface implementation
    class MyRunnable implements Runnable {
        public void run() {
            System.out.println(Thread.currentThread().getName() + " is running");
        }
    }

    // Thread class extension
    class MyThread extends Thread {
        public void run() {
            System.out.println(Thread.currentThread().getName() + " is running");
        }
    }

    public void demonstrateThreading() {
        // Creating and starting a thread using Runnable
        Thread thread1 = new Thread(new MyRunnable(), "RunnableThread");
        thread1.start();

        // Creating and starting a thread by extending Thread class
        MyThread thread2 = new MyThread();
        thread2.setName("ExtendedThread");
        thread2.start();

        // Creating a thread using lambda expression (Java 8+)
        Thread thread3 = new Thread(() -> System.out.println(Thread.currentThread().getName() + " is running"), "LambdaThread");
        thread3.start();

        // Demonstrating thread states
        System.out.println("Thread 3 state: " + thread3.getState());

        // Thread priority
        thread3.setPriority(Thread.MAX_PRIORITY);
        System.out.println("Thread 3 priority: " + thread3.getPriority());

        // Joining threads
        try {
            thread1.join();
            thread2.join();
            thread3.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Demonstrating synchronized block
        synchronized(this) {
            System.out.println(Thread.currentThread().getName() + " in synchronized block");
        }
```

```java
// Using wait() and notify()
        final Object lock = new Object();
        Thread waiter = new Thread(() -> {
            synchronized(lock) {
                try {
                    System.out.println("Waiter is waiting");
                    lock.wait();
                    System.out.println("Waiter is notified");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        Thread notifier = new Thread(() -> {
            synchronized(lock) {
                System.out.println("Notifier is notifying");
                lock.notify();
            }
        });

        waiter.start();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        notifier.start();

        // Demonstrating interrupt
        Thread sleeper = new Thread(() -> {
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                System.out.println("Sleeper was interrupted");
            }
        });
        sleeper.start();
        sleeper.interrupt();

        // Using ThreadLocal
        ThreadLocal<Integer> threadLocal = new ThreadLocal<>();
        threadLocal.set(42);
        System.out.println("ThreadLocal value: " + threadLocal.get());

        // Demonstrating deadlock (be cautious when running this)
        final Object resource1 = new Object();
        final Object resource2 = new Object();
        Thread deadlockThread1 = new Thread(() -> {
            synchronized(resource1) {
                System.out.println("Thread 1: Holding resource 1...");
                try { Thread.sleep(100);} catch (InterruptedException e) {}
                System.out.println("Thread 1: Waiting for resource 2...");
                synchronized(resource2) {
                    System.out.println("Thread 1: Holding resource 1 and resource 2");
                }
            }
        });
        Thread deadlockThread2 = new Thread(() -> {
            synchronized(resource2) {
                System.out.println("Thread 2: Holding resource 2...");
                try { Thread.sleep(100);} catch (InterruptedException e) {}
                System.out.println("Thread 2: Waiting for resource 1...");
                synchronized(resource1) {
                    System.out.println("Thread 2: Holding resource 2 and resource 1");
                }
            }
        });
        deadlockThread1.start();
        deadlockThread2.start();
    }

    public static void main(String[] args) {
        new ThreadingConcepts().demonstrateThreading();
    }
}
```

# DAY 6 (23 September)

```java
// CODE 22
import java.util.HashMap;
import java.util.Map;

public class SimpleCache<K,V> {
    private final Map<K,V> cache;

    public SimpleCache(){
        this.cache = new HashMap<>();
    }
    public void put(K key, V value){
        cache.put(key, value);
    }
    public V get(K key){
        return cache.get(key);
    }
    public void remove(K key){
        cache.remove(key);
    }
    public void clear(){
        cache.clear();
    }

    public int size(){
        return cache.size();
    }
    public static void main(String[] args) {
        SimpleCache<String, String> cache = new SimpleCache<>();
        cache.put("key1", "value1");
        cache.put("key2", "value2");
        cache.put("key3", "value3");
        System.out.println(cache.get("key1"));
        cache.remove("key2");
        System.out.println(cache.size());
        cache.clear();
        System.out.println(cache.size());

    }
}
```

```java
// CODE 23
package com.cache;

import java.util.LinkedHashMap;
import java.util.Map;

public class LRUCache<K,V> extends LinkedHashMap<K,V> {
    private final int capacity;

    public LRUCache(int capacity){
        super(capacity, 0.75f, true);
        this.capacity = capacity;
    }

    @Override
    protected boolean removeEldestEntry(Map.Entry<K,V> eldest){
        return size() > capacity;
    }


    public static void main(String[] args) {
        LRUCache<String, String> cache = new LRUCache<>(3);
        cache.put("key1", "value1");
        cache.put("key2", "value2");
        cache.put("key3", "value3");
        System.out.println(cache);
        cache.get("key2");
        cache.put("key4", "value4");
        cache.put("key5", "value5");
        System.out.println(cache);
    }



}
```

```java
// CODE 24
class BankAccount{
    private volatile int balance=0; //volatile keyword ensures that the value of balance is always read from main memory, not from the CPU
    BankAccount(){
        balance=0;
    }
    public synchronized void deposit(int amount){//synchronized keyword ensures that only one thread can access the method at a time
        balance+=amount;
        System.out.println("Deposited: "+amount+" New Balance: "+balance);
    }
    public synchronized void withdraw(int amount){
        if(balance>=amount){
            balance-=amount;
            System.out.println("Withdrawn: "+amount+" New Balance: "+balance);
        }else{
            System.out.println("Insufficient balance");
        }
    }
    public int getBalance(){
        return balance;
    }
}
public class BankDemo {
    public static void main(String[] args) throws InterruptedException {
        System.out.println("Bank Demo");
        BankAccount account = new BankAccount();
        Thread depositThread = new Thread(() -> {
            for(int i=0;i<5;i++){
                account.deposit(100);

            }
        },"Deposit Thread");
        Thread withdrawThread = new Thread(() -> {
            for(int i=0;i<5;i++){
                account.withdraw(80);
                try{
                    Thread.sleep(100);
                }catch(InterruptedException e){
                    e.printStackTrace();
                }
            }
        },"Withdraw Thread");
        depositThread.start();
        Thread.sleep(10);
        withdrawThread.start();

        try{
            Thread.sleep(2000);
        }catch(InterruptedException e){
            e.printStackTrace();
        }
        System.out.println("Final Balance: "+account.getBalance());
    }
}
```

```
// CODE 25
<dependency>
      <groupId>com.google.guava</groupId>
      <artifactId>guava</artifactId>
      <version>33.3.0-jre</version>
    </dependency>

    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>2.0.16</version>
    </dependency>

    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-simple</artifactId>
      <version>2.0.16</version>
    </dependency>

// ------------------------------------------------------------------------//

package com.cache;
import com.google.common.cache.*;

import java.util.Random;
import java.util.concurrent.TimeUnit;

public class GauvaCacheExample {
    public static void main(String[] args) {
        LoadingCache<String, String> cache = CacheBuilder.newBuilder()
                .maximumSize(200000)
                .expireAfterWrite(10, TimeUnit.MINUTES)
                .build(new CacheLoader<String, String>() {
                    @Override
                    public String load(String key) throws Exception {
                        return "Value for " + key;
                    }
                });
        try{
              String[] genres= {"Action", "Comedy", "Drama", "Horror", "Romance", "Sci-Fi", "Thriller"};
        //Write code to generate 1000 movies with random genres and push in popularMoviewCache
                for(int i=0; i<200000; i++){
                    String movie = "Movie" + i;
                    String genre = genres[new Random().nextInt(genres.length)];
                    cache.put(movie, genre);
                }
                //Thread.sleep(1000);
                for(int i=0;i<100;i++){
                long startTime = System.nanoTime();
                String movie = cache.get("Movie92600");
                long endTime = System.nanoTime();
                System.out.println("Time taken to fetch the movie: " + (endTime - startTime) + " nanoseconds");
                }

                // It would be retrieved from cache
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

```java
// CODE 26

import java.util.HashMap;
import java.util.Map;
import java.util.Random;

class Product {
    private String id;
    private String name;
    private double price;

    public Product(String id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    // Getters and setters omitted for brevity

    @Override
    public String toString() {
        return "Product{id='" + id + "', name='" + name + "', price=" + price + "}";
    }
}

class DatabaseSimulator {
    private Map<String, Product> products = new HashMap<>();
    private Random random = new Random();

    public DatabaseSimulator(int numProducts) {
        for (int i = 0; i < numProducts; i++) {
            String id = "PROD" + i;
            products.put(id, new Product(id, "Product " + i, 10 + random.nextDouble() * 90));
        }
    }

    public Product getProduct(String id) {
        // Simulate database access delay
        try {
            Thread.sleep(100); // 100ms delay to simulate DB access
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return products.get(id);
    }
}


//--------------------------------------------------------------------------------------//

import com.google.common.cache.CacheBuilder;
import com.google.common.cache.CacheLoader;
import com.google.common.cache.LoadingCache;
import java.util.concurrent.TimeUnit;

public class ProductService {
    private final DatabaseSimulator database;
    private final LoadingCache<String, Product> cache;

    public ProductService(int numProducts, int cacheSize) {
        this.database = new DatabaseSimulator(numProducts);
        this.cache = CacheBuilder.newBuilder()
            .maximumSize(cacheSize)
            .expireAfterWrite(10, TimeUnit.MINUTES)
            .recordStats() // This allows us to collect cache statistics
            .build(new CacheLoader<String, Product>() {
                @Override
                public Product load(String id) {
                    return database.getProduct(id);
                }
            });
    }

    public Product getProduct(String id) throws Exception {
        return cache.get(id);
    }

    public void printCacheStats() {
        System.out.println("Cache stats: " + cache.stats());
    }
}

//--------------------------------------------------------------------------------------//

import java.util.Random;

public class CachingPerformanceTest {
    public static void main(String[] args) throws Exception {
        testPerformance(10_000, 1_000); // 10,000 products, 1,000 cache size
        testPerformance(1_000_000, 100_000); // 1 million products, 100,000 cache size
    }

    private static void testPerformance(int numProducts, int cacheSize) throws Exception {
        ProductService service = new ProductService(numProducts, cacheSize);
        Random random = new Random();

        System.out.println("\nTesting with " + numProducts + " products and cache size " + cacheSize);

        // Warm up the cache
        for (int i = 0; i < 1000; i++) {
            service.getProduct("PROD" + random.nextInt(numProducts));
        }

        // Test performance
        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 10000; i++) {
            service.getProduct("PROD" + random.nextInt(numProducts));
        }
        long endTime = System.currentTimeMillis();

        System.out.println("Time taken for 10,000 random product retrievals: " + (endTime - startTime) + "ms");
        service.printCacheStats();
    }
}
```

```java
// CODE 27
package com.example;

import com.google.common.cache.Cache;
import com.google.common.cache.CacheBuilder;
import java.io.*;
import java.nio.file.*;
import java.util.concurrent.TimeUnit;

public class HierarchicalCache {
    private final Cache<String, Product> l1Cache;
    private final Cache<String, Product> l2Cache;
    private final Path l3CacheDir;

    public HierarchicalCache(int l1Size, int l2Size, String l3Path) throws IOException {
        this.l1Cache = CacheBuilder.newBuilder()
                .maximumSize(l1Size)
                .expireAfterWrite(1, TimeUnit.MINUTES)
                .recordStats()
                .build();

        this.l2Cache = CacheBuilder.newBuilder()
                .maximumSize(l2Size)
                .expireAfterWrite(5, TimeUnit.MINUTES)
                .recordStats()
                .build();

        this.l3CacheDir = Paths.get(l3Path);
        Files.createDirectories(l3CacheDir);
    }

    public Product get(String key) throws IOException, ClassNotFoundException {
        // Try L1 Cache
        Product product = l1Cache.getIfPresent(key);
        if (product != null) {
            return product;
        }

        // Try L2 Cache
        product = l2Cache.getIfPresent(key);
        if (product != null) {
            l1Cache.put(key, product);
            return product;
        }

        // Try L3 Cache
        Path filePath = l3CacheDir.resolve(key);
        if (Files.exists(filePath)) {
            try (ObjectInputStream ois = new ObjectInputStream(Files.newInputStream(filePath))) {
                product = (Product) ois.readObject();
                l2Cache.put(key, product);
                l1Cache.put(key, product);
                return product;
            }
        }

        return null;
    }

    public void put(String key, Product value) throws IOException {
        l1Cache.put(key, value);
        l2Cache.put(key, value);

        // Write to L3 Cache
        Path filePath = l3CacheDir.resolve(key);
        try (ObjectOutputStream oos = new ObjectOutputStream(Files.newOutputStream(filePath))) {
            oos.writeObject(value);
        }
    }

    public void printStats() {
        System.out.println("L1 Cache Stats: " + l1Cache.stats());
        System.out.println("L2 Cache Stats: " + l2Cache.stats());
    }
}
```

```java
// CODE 28

import java.util.Random;

public class DataGenerator {
    private static final Random random = new Random();

    public static Product generateProduct(String id) {
        return new Product(id, "Product " + id, 10 + random.nextDouble() * 990);
    }

    public static String generateRandomId(int maxId) {
        return "PROD" + random.nextInt(maxId);
}
}

//------------------------------------------------------------//

package com.example;

import java.io.IOException;

public class HierarchicalCacheTest {
    private static final int TOTAL_PRODUCTS = 100_000;
    private static final int TEST_ITERATIONS = 1_000_000;

    public static void main(String[] args) throws IOException, ClassNotFoundException {
        HierarchicalCache cache = new HierarchicalCache(100, 1000, "l3cache");

        // Populate cache
        System.out.println("Populating cache...");
        for (int i = 0; i < TOTAL_PRODUCTS; i++) {
            String id = "PROD" + i;
            cache.put(id, DataGenerator.generateProduct(id));
        }

        // Test random access
        System.out.println("Testing random access...");
        long startTime = System.currentTimeMillis();

        for (int i = 0; i < TEST_ITERATIONS; i++) {
            String randomId = DataGenerator.generateRandomId(TOTAL_PRODUCTS);
            Product product = cache.get(randomId);
            if (product == null) {
                System.out.println("Product not found: " + randomId);
            }

            if (i % 100000 == 0) {
                System.out.println("Completed " + i + " iterations");
                cache.printStats();
            }
        }

        long endTime = System.currentTimeMillis();
        System.out.println("Total time for " + TEST_ITERATIONS + " random accesses: " + (endTime - startTime) + "ms");
        cache.printStats();
    }
}
```

```java
// CODE 29
import java.util.HashMap;
import java.util.Map;

public class TwoLevelCache {
    private final Map<String, Product> l1Cache;
    private final Map<String, Product> l2Cache;
    private final int l1Capacity;
    private int l1Hits = 0, l2Hits = 0, misses = 0;

    public TwoLevelCache(int l1Capacity, int l2Capacity) {
        this.l1Cache = new HashMap<>(l1Capacity);
        this.l2Cache = new HashMap<>(l2Capacity);
        this.l1Capacity = l1Capacity;
    }

    public Product get(String key) {
        // Check L1 Cache
        if (l1Cache.containsKey(key)) {
            l1Hits++;
            return l1Cache.get(key);
        }

        // Check L2 Cache
        if (l2Cache.containsKey(key)) {
            l2Hits++;
            Product product = l2Cache.get(key);
            // Move to L1 cache
            if (l1Cache.size() >= l1Capacity) {
                // If L1 is full, remove the first entry (simulating LRU)
                String oldestKey = l1Cache.keySet().iterator().next();
                l1Cache.remove(oldestKey);
            }
            l1Cache.put(key, product);
            return product;
        }

        misses++;
        return null; // Not found in either cache
    }

    public void put(String key, Product value) {
        if (l1Cache.size() >= l1Capacity) {
            // If L1 is full, move the first entry to L2 (simulating LRU)
            String oldestKey = l1Cache.keySet().iterator().next();
            l2Cache.put(oldestKey, l1Cache.remove(oldestKey));
        }
        l1Cache.put(key, value);
    }

    public void printStats() {
        int totalRequests = l1Hits + l2Hits + misses;
        System.out.println("Cache Stats:");
        System.out.println("L1 Hits: " + l1Hits + " (" + (l1Hits * 100.0 / totalRequests) + "%)");
        System.out.println("L2 Hits: " + l2Hits + " (" + (l2Hits * 100.0 / totalRequests) + "%)");
        System.out.println("Misses: " + misses + " (" + (misses * 100.0 / totalRequests) + "%)");
    }
}
```

```java
// CODE 30
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;

public class CachingDemo {
    // Simulated "database"
    private static final Map<Integer, String> database = new HashMap<>();

    // Simple cache
    private static final Map<Integer, String> simpleCache = new HashMap<>();

    // LRU cache
    private static final int LRU_CAPACITY = 100;
    private static final LinkedHashMap<Integer, String> lruCache = new LinkedHashMap<Integer, String>(LRU_CAPACITY, 0.75f, true) {
        protected boolean removeEldestEntry(Map.Entry<Integer, String> eldest) {
            return size() > LRU_CAPACITY;
        }
    };

    // Two-level cache
    private static final int L1_CAPACITY = 10;
    private static final int L2_CAPACITY = 100;
    private static final Map<Integer, String> l1Cache = new HashMap<>();
    private static final Map<Integer, String> l2Cache = new HashMap<>();

    // Thread-safe cache
    private static final Map<Integer, String> concurrentCache = new ConcurrentHashMap<>();

    public static void main(String[] args) {
        // Populate the "database"
        for (int i = 0; i < 1000; i++) {
            database.put(i, "Value" + i);
        }

        // Test different caching techniques
        testSimpleCache();
        testLRUCache();
        testTwoLevelCache();
        testConcurrentCache();
    }

    private static void testSimpleCache() {
        System.out.println("\nTesting Simple Cache:");
        long startTime = System.nanoTime();
        for (int i = 0; i < 10000; i++) {
            int key = i % 200; // This will cause some cache hits
            getWithSimpleCache(key);
        }
        long endTime = System.nanoTime();
        System.out.println("Time taken: " + (endTime - startTime) / 1_000_000.0 + " ms");
        System.out.println("Cache size: " + simpleCache.size());
    }

    private static void testLRUCache() {
        System.out.println("\nTesting LRU Cache:");
        long startTime = System.nanoTime();
        for (int i = 0; i < 10000; i++) {
            int key = i % 200; // This will cause some cache hits and evictions
            getWithLRUCache(key);
        }
        long endTime = System.nanoTime();
        System.out.println("Time taken: " + (endTime - startTime) / 1_000_000.0 + " ms");
        System.out.println("Cache size: " + lruCache.size());
    }

    private static void testTwoLevelCache() {
        System.out.println("\nTesting Two-Level Cache:");
        long startTime = System.nanoTime();
        for (int i = 0; i < 10000; i++) {
            int key = i % 200; // This will cause hits in both levels and some misses
            getWithTwoLevelCache(key);
        }
        long endTime = System.nanoTime();
        System.out.println("Time taken: " + (endTime - startTime) / 1_000_000.0 + " ms");
        System.out.println("L1 Cache size: " + l1Cache.size() + ", L2 Cache size: " + l2Cache.size());
    }
```

```java
private static void testConcurrentCache() {
        System.out.println("\nTesting Concurrent Cache:");
        long startTime = System.nanoTime();
        // Simulate concurrent access with multiple threads
        Thread[] threads = new Thread[10];
        for (int t = 0; t < threads.length; t++) {
            threads[t] = new Thread(() -> {
                for (int i = 0; i < 1000; i++) {
                    int key = i % 200;
                    getWithConcurrentCache(key);
                }
            });
            threads[t].start();
        }
        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        long endTime = System.nanoTime();
        System.out.println("Time taken: " + (endTime - startTime) / 1_000_000.0 + " ms");
        System.out.println("Cache size: " + concurrentCache.size());
    }

    private static String getWithSimpleCache(int key) {
        if (!simpleCache.containsKey(key)) {
            simpleCache.put(key, database.get(key));
        }
        return simpleCache.get(key);
    }

    private static String getWithLRUCache(int key) {
        if (!lruCache.containsKey(key)) {
            lruCache.put(key, database.get(key));
        }
        return lruCache.get(key);
    }

    private static String getWithTwoLevelCache(int key) {
        // Check L1 Cache
        if (l1Cache.containsKey(key)) {
            return l1Cache.get(key);
        }
        // Check L2 Cache
        if (l2Cache.containsKey(key)) {
            String value = l2Cache.get(key);
            // Move to L1 cache
            if (l1Cache.size() >= L1_CAPACITY) {
                // If L1 is full, remove the first entry (simulating LRU)
                int oldestKey = l1Cache.keySet().iterator().next();
                l1Cache.remove(oldestKey);
            }
            l1Cache.put(key, value);
            return value;
        }
        // Not in cache, get from database
        String value = database.get(key);
        // Add to L2 cache
        if (l2Cache.size() >= L2_CAPACITY) {
            // If L2 is full, remove the first entry (simulating LRU)
            int oldestKey = l2Cache.keySet().iterator().next();
            l2Cache.remove(oldestKey);
        }
        l2Cache.put(key, value);
        return value;
    }

    private static String getWithConcurrentCache(int key) {
        return concurrentCache.computeIfAbsent(key, k -> database.get(k));
    }
}
```

# DAY 7 (24 September)

```java
// CODE 31
import java.util.concurrent.ConcurrentHashMap;
// import java.util.concurrent.ConcurrentMap;


import java.util.*;

public class BookLibrayCache {
    private final Map<String,Book> bookDatabase=new HashMap<>();
    //Our cache
    private final Map<String,Book> cache=new ConcurrentHashMap<>();

    private final int CACHE_SIZE=5;

    private int cacheHits=0;
    private int cacheMisses=0;

    public BookLibrayCache(){
    bookDatabase.put("1",new Book("1","Book1","Author1"));
    bookDatabase.put("2",new Book("2","Book2","Author2"));
    bookDatabase.put("3",new Book("3","Book3","Author3"));
    bookDatabase.put("4",new Book("4","Book4","Author4"));
    bookDatabase.put("5",new Book("5","Book5","Author5"));
    bookDatabase.put("6",new Book("6","Book6","Author6"));
    }

    public Book getBook(String bookId){
        Book book=cache.get(bookId);
        if(book!=null){
            cacheHits++;
            System.out.println("Cache Hit for bookId: "+bookId);
            return book;
        }
        else{
            cacheMisses++;
            System.out.println("Cache Miss for bookId: "+bookId);
            book=bookDatabase.get(bookId);
            if(book!=null){
                addToCache(bookId,book);
            }
            return book;
        }
    }
    private void addToCache(String bookId,Book book){
        if(cache.size()>CACHE_SIZE){
            String keyToRemove=cache.keySet().iterator().next();
            cache.remove(keyToRemove);
            System.out.println("Cache is full. Removing least recently used bookId: "+keyToRemove);
        }
        cache.put(bookId,book);
        System.out.println("Added to cache: "+bookId);
    }
    public void printCacheStatistics(){
        System.out.println("Cache Hits: "+cacheHits);
        System.out.println("Cache Misses: "+cacheMisses);
        System.out.println("Cache Hit Ratio: "+((double)cacheHits/(cacheHits+cacheMisses)));
        System.out.println("Current Cache Size: "+cache.size());
        System.out.println("Books in Cache: "+cache.keySet());
    }
    private static class Book{
        private String id;
        private String title;
        private String author;

        public Book(String id,String title,String author){
            this.id=id;

        }
        @Override
        public String toString(){
            return "Book{id='"+id+"', title='"+title+"', author='"+author+"'}";
        }
    }
    public static void main(String[] args){
        BookLibrayCache cache=new BookLibrayCache();
        String []requestedBooks={"1","2","3","4","5","6","1","2","3","4","5","6","5","4","3","9"};
        for(String bookId:requestedBooks){

            Book book=cache.getBook(bookId);
            if(book!=null){
                System.out.println("Retived book: "+book);
            }
            else{
                System.out.println("Book not found : "+book);
            }
            System.out.println("----------------------------------");

        }
        cache.printCacheStatistics();
    }
}
```

```java
// CODE 32

import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicInteger;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.io.Serializable;
import java.io.ObjectInputStream;
import java.io.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
//Write comment for each line of code auto generated why its is required
public class DetailDocumentCache {
    //ConcurrentHashMap is used instead of hashmap because it is thread safe and it is used in concurrent environment
    //It allows multiple threads to access and modify the cache concurrently without causing any issues
    //Unlike Collections.synchronizedMap, it does not lock the entire map during write operations,
    //better performance and scalability in multi-threaded environments
    private final ConcurrentHashMap<String,Document> cache;

    private final String diskStoragePath;
    private final int maxCacheSize;

    //AutomicInteger is used instead of Integer because it provides atomic operations on the integer value
    //AtomicInteger ensures that the integer value is updated atomically, which means that the value is updated in a single step without any interruptions
    //This is useful in concurrent environments where multiple threads may be accessing and updating the integer value simultaneously
    //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
    private final AtomicInteger cacheHits=new AtomicInteger(0);
    //Why not volatile instead of AtomicInteger?
    //Volatile keyword is used to ensure that the value of the variable is always read from the main memory and not from the cache
    //It ensures that the variable is always up to date and visible to all threads
    //However, it does not provide atomic operations on the variable
    //Volatile keyword is useful when you want to ensure that the variable is always read from the main memory and not from the cache
    //However, it does not provide atomic operations on the variable

    //What is atomic can you give an example?
    //Atomic operations are operations that are performed in a single step and cannot be interrupted by other threads
    //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
    private final AtomicInteger cacheMisses=new AtomicInteger(0);

    public DetailDocumentCache(int maxCacheSize,String diskStoragePath){
        this.maxCacheSize=maxCacheSize;
        this.diskStoragePath=diskStoragePath;
        this.cache=new ConcurrentHashMap<>(maxCacheSize);

        try{
            Files.createDirectories(Paths.get(diskStoragePath));
        }catch(IOException e){
            e.printStackTrace();
        }
    }

    public Document getDocument(String documentId) throws IOException,ClassNotFoundException{
        Document cachedDocument=cache.get(documentId);
        if(cachedDocument!=null){
            cacheHits.incrementAndGet();
            return cachedDocument;
        }
        else{
            cacheMisses.IncrementAndGet();
            Document document=loadDocumentFromDisk(documentId);
            if(document!=null){
                addToCache(documentId,document);
               // evictCacheIfNecessary();
            }
            return document;

        }

    }
```

```java
public void saveDocument(Document document) throws IOException{
        cache.put(document.getDocumentId(),document);
        evictCacheIfNecessary();
        saveToDisk(document);
    }
    private void saveToDisk(Document document) throws IOException{
        Path filePath=Paths.get(diskStoragePath,document.getDocumentId());
        try(ObjectOutputStream out=new ObjectOutputStream(Files.newOutputStream(filePath))){
            out.writeObject(document);
        }
    }
    public void printCacheStatistics(){
        System.out.println("Cache Hits: "+cacheHits.get());
        System.out.println("Cache Misses: "+cacheMisses.get());
        System.out.println("Cache Size: "+cache.size());
        System.out.println("Cache Capacity: "+maxCacheSize);
        System.out.println("Cache Efficiency: "+((double)cacheHits.get()/(cacheHits.get()+cacheMisses.get()))*100)+"%");
        System.out.println("Cache Hit Ratio: "+((double)cacheHits.get()/(cacheHits.get()+cacheMisses.get()))*100)+"%");
        System.out.println("Cache Miss Ratio: "+((double)cacheMisses.get()/(cacheHits.get()+cacheMisses.get()))*100)+"%");

    }
    private void addToCache(String documentId,Document document){
        if(cache.size()>maxCacheSize){
            evictCacheIfNecessary();
        }
        cache.put(documentId,document);
    }
    private Document loadDocumentFromDisk(String documentId) throws IOException,ClassNotFoundException{
        Path filePath=Paths.get(diskStoragePath,documentId);
        if(Files.exists(filePath)){
            try(ObjectInputStream in=new ObjectInputStream(Files.newInputStream(filePath))){
                return (Document)in.readObject();
            }
        }
        return null;
    }
    private void evictCacheIfNecessary(){
        while(cache.size()>maxCacheSize){
            String oldestDocument =cache.keySet().iterator().next();
            cache.remove(oldestDocument);
        }
    }
    //Why static class instead of inner class?
    //Static class can be instantiated without having to instantiate the outer class
    //Static class can be used in the outer class without having to instantiate the outer class
    //Why to implemont serializable interface?
    //Serializable interface is implemented to allow the Document object to be converted to a byte stream and stored in the file system
    //This allows the Document object to be saved to the file system and loaded back into the program
    public static class Document implements Serializable{
        private final String documentId;
        private final String content;
        private final long timestamp;
        private final long lastModifiedTime;

        public Document(String documentId,String content,long timestamp,long lastModifiedTime){
            this.documentId=documentId;
            this.content=content;
            this.timestamp=timestamp;
            this.lastModifiedTime=lastModifiedTime;
        }
        public String getDocumentId(){
            return documentId;
        }
        public String getContent(){
            return content;
        }
        public long getTimestamp(){
            return timestamp;
        }
        public long getLastModifiedTime(){
            return lastModifiedTime;
        }
        @Override
        public String toString(){
            return "Document{"+"documentId='"+documentId+'\''+", content='"+content+'\''+", timestamp="+timestamp+", lastModifiedTime="+lastModifiedTime+'}';
        }

    }
```

```java
public static void main(String[] args){
        DetailDocumentCache cache=new DetailDocumentCache(100,"C:\\Codebase\\my-caching-project\\my-caching-project\\src\\main\\java\\com\\cache\\boooksystem\\diskStorage");
        //How ExecutorService is better than Thread creating by extrending thread class?
        //ExecutorService is better than Thread creating by extrending 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(2);
        //Simulate multiple threads accessing and modifying the document
System.err.println("Starting the simulation");
        //Explain these code in detail
        //Simulate multiple threads accessing and modifying the document
        for(int i=0;i<100;i++){
        final int index=i;
        executor.execute(()->{
            try{
                //Explain these code in detail
                //Simulate a document with a unique ID based on the index
                //Explain these code in detail
                //Simulate a document with a unique ID based on the index
                // Explain if and else condition in detail
                //If the index is divisible by 10, a new document is created and saved to the cache
                //If the index is not divisible by 10, the document is retrieved from the cache
                //Explain the purpose of the try and catch block
                //The try and catch block is used to catch any exceptions that may be thrown by the code
                //The try block is used to execute the code that may throw an exception
                //The catch block is used to catch any exceptions that may be thrown by the code
                //Explain the purpose of the if and else condition
                //The if and else condition is used to check if the index is divisible by 10
                //If the index is divisible by 10, a new document is created and saved to the cache
                //If the index is not divisible by 10, the document is retrieved from the cache
                String id = "Document"+ (index);
                if(index%10==0){
                    Document newDocument=new Document(id,"Content for "+id,System.currentTimeMillis(),System.currentTimeMillis());
                    cache.saveDocument(newDocument);
                    System.out.println("Saved document: "+id);

                }else{
                    Document document=cache.getDocument(id);
                    if(document!=null){
                        System.out.println("Retrieved document: "+id);
                    }else{
                        System.out.println("Document not found: "+id);
                    }
                }
            }catch(IOException | ClassNotFoundException e){
                e.printStackTrace();
            }
            finally{
                System.out.println("Thread "+index+" completed");
               // executor.shutdown();
                try{
                    executor.awaitTermination(1,TimeUnit.MINUTES);
                }catch(InterruptedException e){
                    e.printStackTrace();
                }

            }
        });
        cache.printCacheStatistics();
    }
}
}
```

```java
// CODE 33

import java.util.*;
import java.io.*;
import java.util.concurrent.*;
public class BookLibrarySystem {
    public static void main(String[] args) {
        BookLibrary bookLibrary=new BookLibrary();
        bookLibrary.addBook(new Book("123","Harry Potter","J.K.Rowling"));
        bookLibrary.addBook(new Book("456","To Kill a Mockingbird","Harper Lee"));
        bookLibrary.addBook(new Book("789","1984","George Orwell"));
        bookLibrary.addBook(new Book("101","Pride and Prejudice","Jane Austen"));
        bookLibrary.addBook(new Book("102","The Great Gatsby","F. Scott Fitzgerald"));
        bookLibrary.addBook(new Book("103","Moby Dick","Herman Melville"));
        bookLibrary.addBook(new Book("104","War and Peace","Leo Tolstoy"));
        bookLibrary.addBook(new Book("105","Hamlet","William Shakespeare"));
        bookLibrary.addBook(new Book("106","Macbeth","William Shakespeare"));

        System.out.println(bookLibrary.getAllBooks());
        System.out.println(bookLibrary.getAuthors());
        System.out.println(bookLibrary.getBookCountByAuthor());

        // How Future is used in Asynchronous programming:
        // Future represents the result of an asynchronous computation. It provides methods to check if the computation is complete,
        // to wait for its completion, and to retrieve the result of the computation.

        // Explain the difference between Future and ExecutorService:
        // Future represents a single asynchronous computation, while ExecutorService is a higher-level facility
        // for executing and managing multiple asynchronous tasks.

        Future<Book> futureMostPopularBook=bookLibrary.getMostPopularBookAsync();//
        try{
            Book mostPopularBook=futureMostPopularBook.get();
            System.out.println("Most popular book: "+mostPopularBook.toString());
        }catch(InterruptedException | ExecutionException e){
            e.printStackTrace();
        }finally{
            bookLibrary.executorService.shutdown();
        }
        System.out.println("Program continues while waiting for the most popular book");
    }
    static class BookLibrary{
        private List<Book> books = new ArrayList<>();

        private Set<String> authors = new HashSet<>();

        private Map<String,Integer> bookCountByAuthor = new HashMap<>();

        private ExecutorService executorService = Executors.newSingleThreadExecutor();
        BookLibrary(){
            books=new ArrayList<>();
            authors=new HashSet<>();
            bookCountByAuthor=new HashMap<>();
            executorService=Executors.newSingleThreadExecutor();
        }
        public void addBook(Book book){
            books.add(book);
            authors.add(book.author);
            bookCountByAuthor.put(book.author,bookCountByAuthor.getOrDefault(book.author,0)+1);
        }
        public List<Book> getAllBooks(){
            return new ArrayList<>(books);
        }
        public Set<String> getAuthors(){
            return new HashSet<>(authors);
        }
        public Map<String,Integer> getBookCountByAuthor(){
            return new HashMap<>(bookCountByAuthor);
        }
        //Asynchronous programming
        public Future<Book> getMostPopularBookAsync(){
        return executorService.submit(()->{
            Thread.sleep(2000);

            return books.isEmpty()?null:getMostPopularBook();
        });
    }
    private Book getMostPopularBook(){
        return books.get(0);
    }
    }
    static class Book implements Serializable{
        private String isbn;
        private String title;
        private String author;
        public Book(String isbn,String title,String author){
            this.isbn=isbn;
            this.title=title;
            this.author=author;
        }
        // trainsent feild, it would not be serialized
        transient int curentPage=0;//transient keyword is used to indicate that a field should not be serialized
        @Override
        public String toString(){
            return "Book{"+"isbn='"+isbn+'\''+", title='"+title+'\''+", author='"+author+'\''+'}';
        }
    }
}
```

# Group Activity:

Developed a MOVIE CACHE SYSTEM in Java.
The code is attached below.

Group:

1. Ashish
2. Laasya
3. Karthikey
4. Gitesh

```java
// CODE 34
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

class MediaContent {
    private String title;
    private String genre;
    private int duration; // duration in minutes

    public MediaContent(String title, String genre, int duration) {
        this.title = title;
        this.genre = genre;
        this.duration = duration;
    }

    public String getTitle() {
        return title;
    }

    public String getGenre() {
        return genre;
    }

    public int getDuration() {
        return duration;
    }

    @Override
    public String toString() {
        return "Title: " + title + ", Genre: " + genre + ", Duration: " + duration + " minutes";
    }
}

class UserCache {
    private HashMap<String, MediaContent> mediaCache;
    private final int MAX_CACHE_SIZE = 5;

    public UserCache() {
        mediaCache = new HashMap<>();
    }

    public MediaContent getMediaFromCache(String title) {
        return mediaCache.getOrDefault(title, null);
    }

    public void addMediaToCache(MediaContent media) {
        if (mediaCache.size() >= MAX_CACHE_SIZE) {
            // Simple eviction strategy: remove the first entry (FIFO)
            String oldestTitle = mediaCache.keySet().iterator().next();
            mediaCache.remove(oldestTitle);
        }
        mediaCache.put(media.getTitle(), media);
    }

    public boolean isInCache(String title) {
        return mediaCache.containsKey(title);
    }
}

class User {
    private String username;
    private String subscriptionPlan;
    private UserCache userCache; // Each user has their own cache

    public User(String username, String subscriptionPlan) {
        this.username = username;
        this.subscriptionPlan = subscriptionPlan;
        this.userCache = new UserCache(); // Initialize a cache for each user
    }

    public String getUsername() {
        return username;
    }

    public String getSubscriptionPlan() {
        return subscriptionPlan;
    }

    public UserCache getUserCache() {
        return userCache;
    }
}
```

```java
class MediaService {
    public MediaContent fetchMedia(User user, String title) {
        // Check if the media is in the user's cache
        if (user.getUserCache().isInCache(title)) {
            System.out.println("Fetching '" + title + "' from " + user.getUsername() + "'s cache.");
            return user.getUserCache().getMediaFromCache(title);
        }

        // Simulate a database or API call to fetch media content
        System.out.println("Fetching '" + title + "' from database/API for " + user.getUsername() + ".");
        MediaContent media = fetchMediaFromDatabase(title);

        // Add the media to the user's cache
        if (media != null) {
            user.getUserCache().addMediaToCache(media);
        }
        return media;
    }

    private MediaContent fetchMediaFromDatabase(String title) {
        // Simulate data source with hardcoded media list
        List<MediaContent> mediaList = new ArrayList<>();
        mediaList.add(new MediaContent("Stranger Things", "Sci-Fi", 50));
        mediaList.add(new MediaContent("The Crown", "Drama", 60));
        mediaList.add(new MediaContent("The Office", "Comedy", 30));
        mediaList.add(new MediaContent("Money Heist", "Thriller", 45));
        mediaList.add(new MediaContent("Breaking Bad", "Crime", 55));
        mediaList.add(new MediaContent("Friends", "Comedy", 30));

        // Find media by title
        for (MediaContent media : mediaList) {
            if (media.getTitle().equalsIgnoreCase(title)) {
                return media;
            }
        }
        return null;
    }
}

public class Media {
    public static void main(String[] args) {
        MediaService mediaService = new MediaService();

        // Create two users, each with their own cache
        User user1 = new User("john_doe", "Premium");
        User user2 = new User("jane_doe", "Basic");

        // Simulate user1 requests
        System.out.println(user1.getUsername() + " is watching:");
        mediaService.fetchMedia(user1, "Stranger Things");
        mediaService.fetchMedia(user1, "Money Heist");

        // Fetch again for user1 to trigger cache
        System.out.println("\nSecond request (should be fetched from user1's cache):");
        mediaService.fetchMedia(user1, "Stranger Things");

        // Simulate user2 requests
        System.out.println("\n" + user2.getUsername() + " is watching:");
        mediaService.fetchMedia(user2, "Breaking Bad");
        mediaService.fetchMedia(user2, "Friends");

        // Fetch again for user2 to trigger cache
        System.out.println("\nSecond request (should be fetched from user2's cache):");
        mediaService.fetchMedia(user2, "Breaking Bad");
    }
}
```

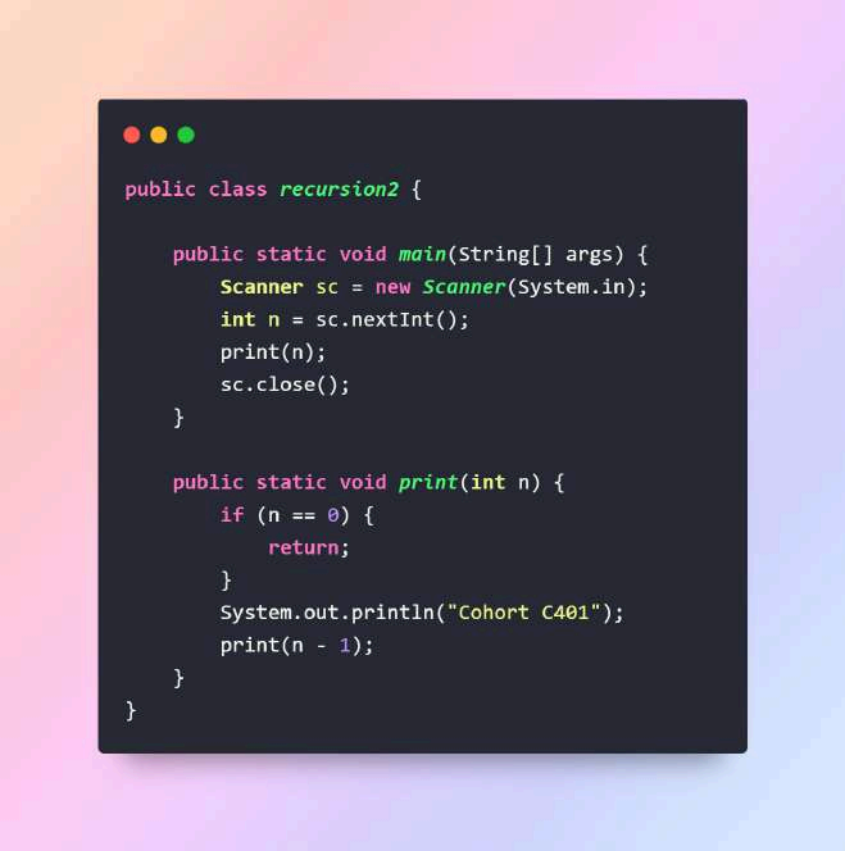
# DAY 8 (25 September)

```java
// CODE 35
package com.cache;
import java.io.Serializable;
import java.util.concurrent.ConcurrentHashMap;
import java.util.PriorityQueue;
import java.util.Comparator;


public class DocumentCache {
    // Thread-safe map to store documents
    private final ConcurrentHashMap<String, Document> cache;
    // Queue to manage document eviction based on expiration and frequency
    private final PriorityQueue<Document> evictionQueue;
    private final int maxCacheSize;

    public DocumentCache(int maxCacheSize) {
        this.cache = new ConcurrentHashMap<>();
        // Custom comparator for eviction priority
        this.evictionQueue = new PriorityQueue<>(
            Comparator.comparingLong(Document::getExpirationTime)
                      .thenComparingInt(Document::getFrequency)
        );
        this.maxCacheSize = maxCacheSize;
    }
```

```java
// Document class representing cached items
    public static class Document implements Serializable {
        private final String documentId;
        private final String content;
        private final long ttl; // Time-To-Live in milliseconds
        private long expirationTime;
        private int frequency;

        public Document(String documentId, String content, long ttl) {
            this.documentId = documentId;
            this.content = content;
            this.ttl = ttl;
            this.expirationTime = System.currentTimeMillis() + ttl;
            this.frequency = 0;
        }

        public long getExpirationTime() { return expirationTime; }
        public int getFrequency() { return frequency; }

        // Update document access stats
        public void updateAccess() {
            this.expirationTime = System.currentTimeMillis() + ttl;
            this.frequency++;
        }
    }

    // Add a new document to the cache
    public void addDocument(Document document) {
        cache.put(document.documentId, document);
        evictionQueue.offer(document);
        evictIfNecessary();
    }

    // Retrieve a document from the cache
    public Document getDocument(String documentId) {
        Document document = cache.get(documentId);
        if (document != null && !isExpired(document)) {
            // Update document stats and reposition in eviction queue
            evictionQueue.remove(document);
            document.updateAccess();
            evictionQueue.offer(document);
            return document;
        } else {
            // Remove expired or non-existent document
            cache.remove(documentId);
            evictionQueue.remove(document);
            return null;
        }
    }

    // Check if a document has expired
    private boolean isExpired(Document document) {
        return System.currentTimeMillis() > document.getExpirationTime();
    }

    // Evict documents if cache is full or contains expired items
    private void evictIfNecessary() {
        while (!evictionQueue.isEmpty() &&
               (isExpired(evictionQueue.peek()) || cache.size() > maxCacheSize)) {
            Document toEvict = evictionQueue.poll();
            cache.remove(toEvict.documentId);
        }
    }

    // Manually trigger cleanup of expired documents
    public void removeExpiredDocuments() {
        long currentTime = System.currentTimeMillis();
        evictionQueue.removeIf(doc -> {
            if (currentTime > doc.getExpirationTime()) {
                cache.remove(doc.documentId);
                return true;
            }
            return false;
        });
    }
}
```

# DATA STRUCTURES AND ALGORITHM

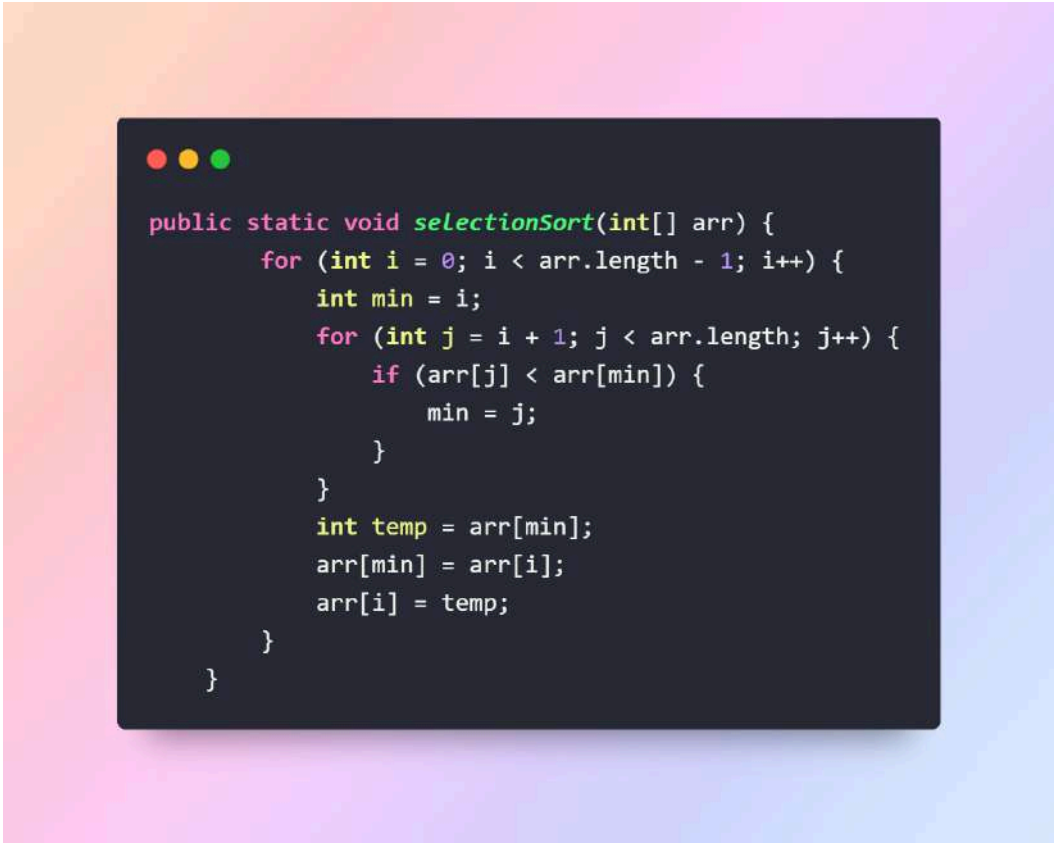
```java
class KadanesAlgo {
    public static void main(String[] args) {
        int[] arr = {-2,1,-3,4,-1,2,1,-5,4};
        System.out.println(maxSubArray(arr));
    }

    public static int maxSubArray(int[] arr) {
        int maxSum = arr[0];
        int currentSum = arr[0];
        for (int i = 1; i < arr.length; i++) {
            currentSum = Math.max(arr[i], currentSum + arr[i]);
            maxSum = Math.max(maxSum, currentSum);
        }
        return maxSum;
    }

}
```



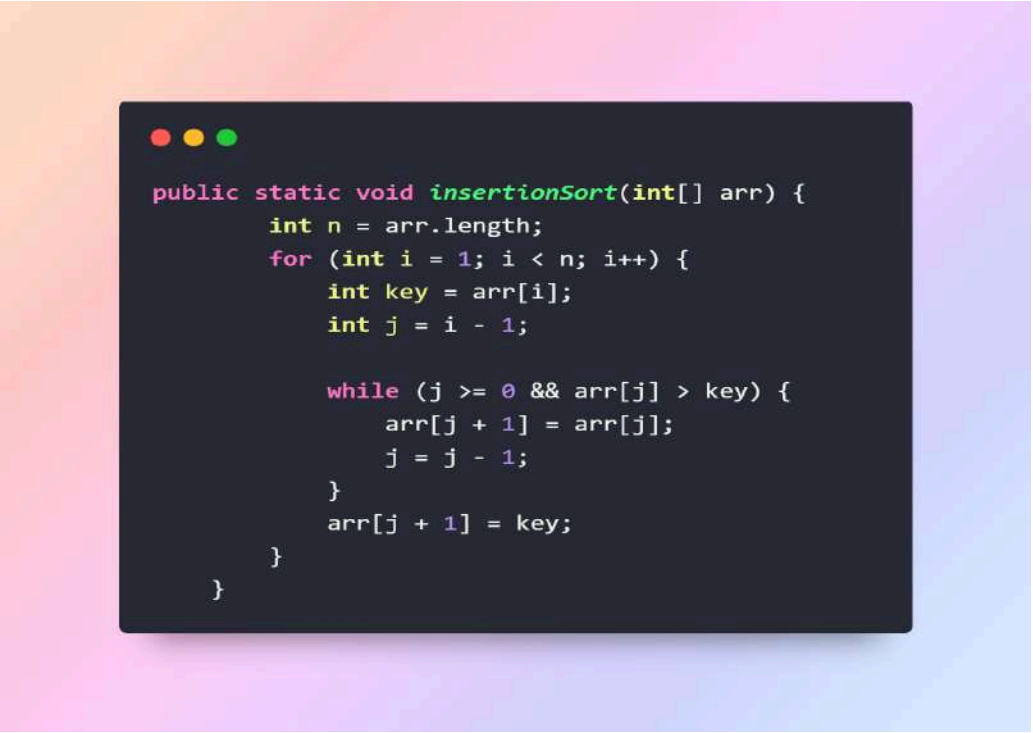
```java
public class recursion2 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        print(n);
        sc.close();
    }

    public static void print(int n) {
        if (n == 0) {
            return;
        }
        System.out.println("Cohort C401");
        print(n - 1);
    }
}
```

# SORTING ALGORITHMS:

```java
public static void bubbleSort2(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = 0; j < arr.length - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }
```



```java
public static void selectionSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            int min = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[min]) {
                    min = j;
                }
            }
            int temp = arr[min];
            arr[min] = arr[i];
            arr[i] = temp;
        }
    }
```



```java
public static void insertionSort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int key = arr[i];
            int j = i - 1;

            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
    }
```

## DAY 9 (26 September)

```java
public static int findSecondLargest(int[] arr) {
        if (arr.length < 2) {
            return -1;
        }
        int largest = 0;
        int secondLargest = 0;
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > arr[largest]) {
                secondLargest = largest;
                largest = i;
            } else if (arr[i] > arr[secondLargest] && arr[i] < arr[largest]) {
                secondLargest = i;
            }
        }
        return arr[secondLargest];
    }
```

```java
public static int[] removeDuplicates2(int[] arr) {
        if (arr == null || arr.length == 0) {
            return new int[0];
        }

        int[] result = new int[arr.length];
        int index = 0;
        result[index++] = arr[0];  // Add the first element

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] != arr[i - 1]) {
                result[index++] = arr[i];
            }
        }

        // Trim the result array to the correct size
        return Arrays.copyOf(result, index);
    }
```

```java
static void moveZerosToEnd(int[] arr) {
        int index = 0;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] != 0)
                arr[index] = arr[i];
                index++;
        }
        while (index < arr.length)
            arr[index++] = 0;
    }
```

```java
public List<Integer> findDuplicates(int[] nums)
    {
        int n = nums.length; // Get the length of the input
                             // array

        int[] arr = new int[n]; // Create an auxiliary
                                // array of size n+1
        List<Integer> list
            = new ArrayList<>(); // List to store duplicates

        // Iterate through each number in the input array
        for (int i : nums) {
            if (++arr[i] > 1) // Increment the count for
                // this number and check if it is now a duplicate

                list.add(i); // If it is a duplicate, add it
                             // to the list
        }

        return list; // Return the list of duplicates
    }
```

```java
static void reverseArray(int[] arr) {

        // Initialize left to the beginning and right to the end
        int left = 0, right = arr.length - 1;

        // Iterate till left is less than right
        while (left < right) {

            // Swap the elements at left and right position
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;

            // Increment the left pointer
            left++;

            // Decrement the right pointer
            right--;
        }
    }
```

```java
public static int missingNumber(int n, int[] arr)
    {
        int sum = 0;

        // Calculate the sum of array elements
        for (int i = 0; i < n - 1; i++) {
            sum += arr[i];
        }

        // Calculate the expected sum
        int expectedSum = (n * (n + 1)) / 2;

        // Return the missing number
        return expectedSum - sum;
    }
```

```java
// LEFT ROTATE AN ARRAY BY ONE
    static void rotate()
    {
        int last_el = arr[arr.length - 1], i;
        for (i = arr.length - 1; i > 0; i--)
            arr[i] = arr[i - 1];
        arr[0] = last_el;
    }
```

```java
// REVERSAL ALGORITHM
static void leftRotate(int arr[], int d)
    {

        if (d == 0)
            return;

        int n = arr.length;
        // in case the rotating factor is
        // greater than array length
        d = d % n;
        reverseArray(arr, 0, d - 1);
        reverseArray(arr, d, n - 1);
        reverseArray(arr, 0, n - 1);
    }

    /*Function to reverse arr[] from index start to end*/
    static void reverseArray(int arr[], int start, int end)
    {
        int temp;
        while (start < end) {
            temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
            start++;
            end--;
        }
    }
```

```java
// Returns index of x if it is present in arr[].
    int binarySearch(int arr[], int x)
    {
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;

            // Check if x is present at mid
            if (arr[mid] == x)
                return mid;

            // If x greater, ignore left half
            if (arr[mid] < x)
                low = mid + 1;

            // If x is smaller, ignore right half
            else
                high = mid - 1;
        }

        // If we reach here, then element was
        // not present
        return -1;
    }
```

# DAY 10 (27 September)

# Group Activity:

Developed a Employee Hiring Management System in Java.
The code is attached below.

Group:
1. Ashish
2. Anushka
3. Lalitha
4. Maanvitha

## Project Title:

Employee Hiring Management System Using Cache and Database Integration

## Introduction:

This project is an Employee Hiring Management System designed to manage the hiring process between companies and employees. It integrates caching concepts for efficient data retrieval and connects to a MySQL database to store employee and company data. The system shortlists employees based on their skills and work experience, matching company requirements, while utilizing cache to enhance performance.

## Project Objective:

The primary objective of this system is to manage the process of employee recruitment by allowing:

- Employees to input their skills and experience.
- Companies to specify job requirements.
- Both employees and companies to find suitable matches based on set criteria.

Additionally, the system uses caching to store frequently accessed data, improving the system's response time for repeated queries.

## Key Features:

1. Employee Management:
    - Employees can enter their details, including skills and years of experience.
    - The system stores employee information in a MySQL database.
    - Employees can query the system to find companies they are eligible to apply to based on their skills and experience.
2. Company Management:
    - Companies can input job requirements, specifying the required skills and minimum experience.
    - Company information is stored in a MySQL database.
    - Companies can search for potential employees that meet their criteria.
3. Caching:
    - The system caches employee data based on frequently required skills.
    - This reduces database access and improves the response time when the same queries are executed multiple times.
    - A skill frequency tracker records how often a specific skill is requested by companies.
4. Database Integration:
    - Employees and companies are stored in two MySQL tables (`employees` and `companies`).
    - CRUD (Create, Read, Update, Delete) operations are performed on these tables to manage the employee and company data.
5. User Interface:
    - A simple command-line interface allows employees and companies to interact with the system.
    - The user selects options to add data, search for eligible companies or employees, and view cached data.

## System Architecture:

1. Employee Class:
    - Contains attributes such as `name`, `skills`, and `workExperience`.
    - Represents the details of each employee.
2. Company Class:
    - Contains attributes such as `name`, `requiredSkill`, and `minExperience`.
    - Represents the details of a company's job requirements.
3. Main Project Class:
    - Handles all user inputs and logic flow.
    - Manages the connection to the MySQL database using JDBC.
    - Implements caching to store employee details based on frequently requested skills.

## How the System Works:

1. Database Connection:
    - When the application starts, a connection to the MySQL database is established using the JDBC API.
    - All employee and company information is stored in respective tables, ensuring persistence.
2. Employee Operations:
    - The employee provides their name, skills (comma-separated), and years of experience.
    - This data is stored in the `employees` table.
    - When the employee requests eligible companies, the system retrieves their skills and experience, then queries the database for matching company requirements.
3. Company Operations:
    - The company provides job requirements, including required skills and minimum experience.
    - This data is stored in the `companies` table.
    - When the company searches for employees, the system queries the database to find employees who match their requirements based on skills and experience.
4. Caching:
    - For frequently requested skills, employee data is stored in an in-memory cache (using HashMaps).
    - If a company requests employees with a specific skill multiple times, the system will fetch the data from the cache instead of querying the database, thus speeding up the response.
    - A frequency counter tracks how often each skill is requested.
5. Cache Viewing:
    - The system provides an option to view all cached employee data, categorized by the skill and the number of companies requesting it.
6. Closing Connections:
    - When the application is closed, all open resources such as database connections and result sets are properly closed to avoid memory leaks.

## Advantages of the System:

- Efficiency: By using caching, the system can respond faster to frequent queries, reducing the load on the database.

- Flexibility: Companies can specify detailed job requirements, and employees can view eligible companies based on their skill set.
- Scalability: The use of a database allows the system to handle large volumes of employee and company data.

## Conclusion:

The Employee Hiring Management System simplifies the recruitment process by matching employees with suitable companies based on skill requirements and work experience. By integrating caching mechanisms and a robust database system, it provides an efficient solution for handling multiple queries and large datasets. This project demonstrates the practical application of cache and database integration in a real-world scenario.

## Database Schema:

```
show databases;
create database project;
use project;
CREATE TABLE employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50),
    skills VARCHAR(255),
    work_experience INT
);

CREATE TABLE companies (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    required_skill VARCHAR(50),
    min_experience INT
);

insert into employees( id, first_name, skills, work_experience)
values
(101, 'ashish', 'java', 3),
(102, 'maanvitha', 'python', 5),
(103, 'lalitha', 'C++', 2),
(104, 'anushka', 'SQL', 4);

select * from employees;
insert into companies( id, name, required_skill, min_experience)
values
(999, 'TCS', 'java', 1),
(888, 'INFOSYS', 'python', 3);

select * from companies;
```

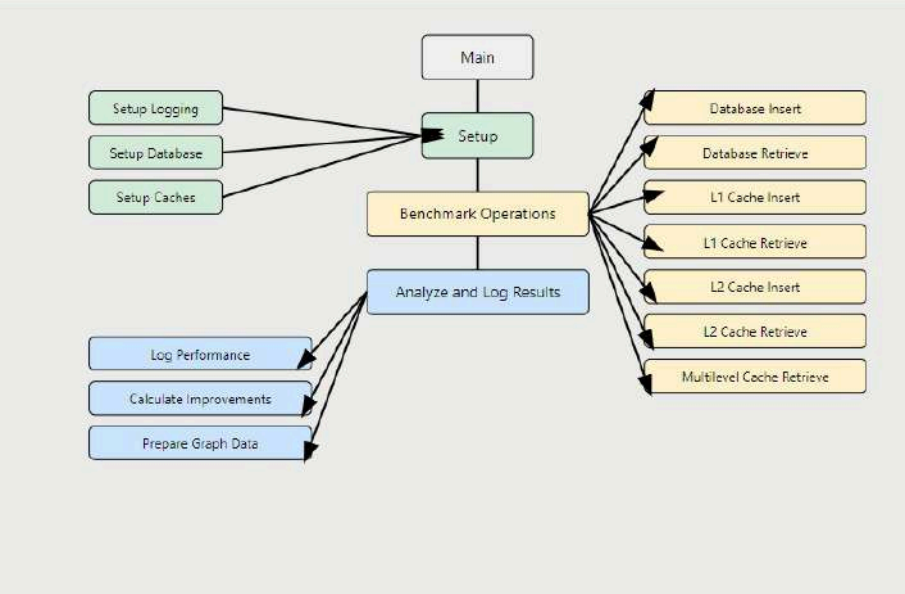
## Project (code implementation):

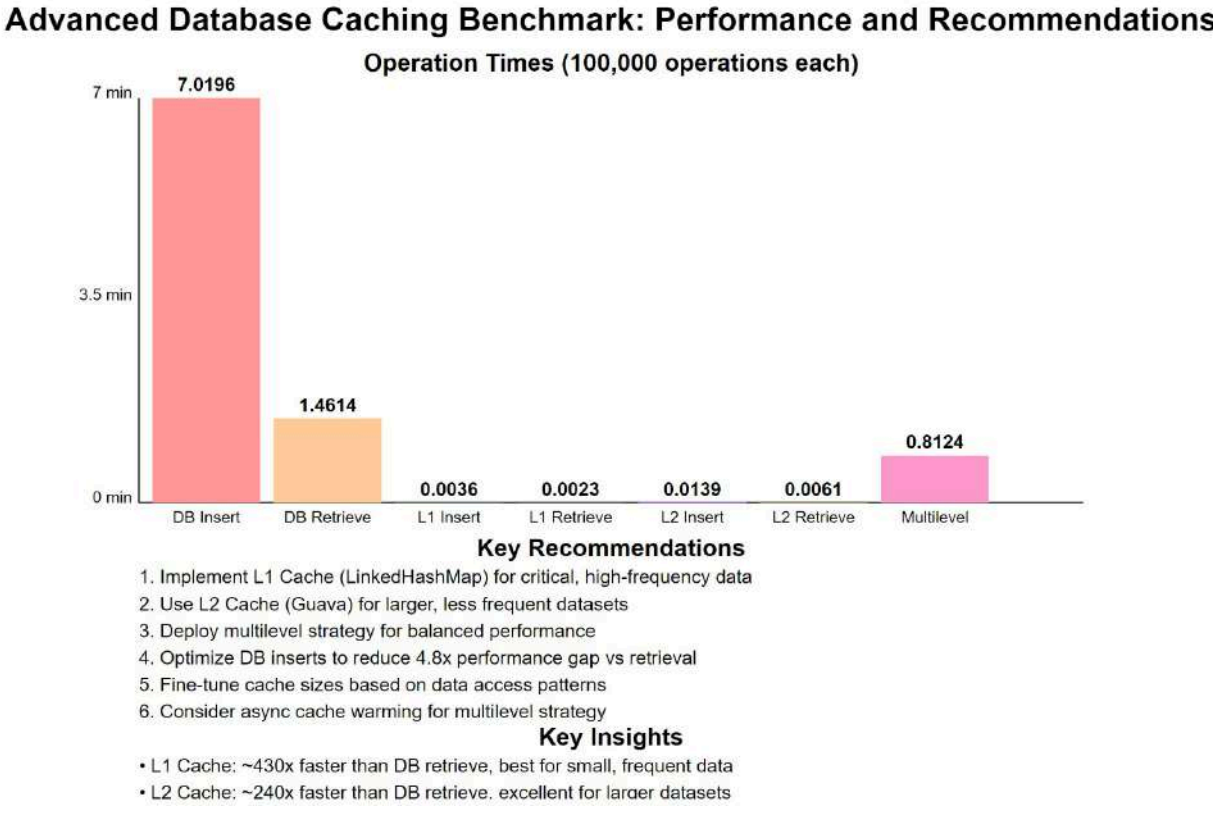
```java
import java.sql.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

// User class to represent an employee
class User {
    String name;
    String skills;
    int workExperience;

    // Constructor for User class
    User(String name, String skills, int workExperience) {
        this.name = name;
        this.skills = skills;
        this.workExperience = workExperience;
    }
}

// Company class to represent a company
class Company {
    String name;
    String requiredSkill;
    int minExperience;

    // Constructor for Company class
    Company(String name, String requiredSkill, int minExperience) {
        this.name = name;
        this.requiredSkill = requiredSkill;
        this.minExperience = minExperience;
    }
}

// Main class for the Job Portal project
public class Project {
    // ArrayLists to store User and Company objects
    static ArrayList<User> users = new ArrayList<>();
    static ArrayList<Company> companies = new ArrayList<>();
    // Scanner object for user input
    static Scanner sc = new Scanner(System.in);

    // Cache to store employees based on skills
    static Map<String, ArrayList<User>> employeeCache = new HashMap<>();
    // Map to store frequency of required skills
    static Map<String, Integer> skillFrequency = new HashMap<>();

    // Database connection details
    private static final String URL = "jdbc:mysql://localhost:3306/project";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "root";

    // Database connection objects
    private static Connection connection;
    private static PreparedStatement preparedStatement;
    private static ResultSet resultSet;

    // Main method - entry point of the program
    public static void main(String[] args) {
        try {
            // Establish database connection
            connection = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            // Preload data (if any)
            preloadData();

            // Main program loop
            boolean exit = false;
            while (!exit) {
                // Display main menu
                System.out.println("\n*** Job Portal ***");
                System.out.println("1. Employee");
                System.out.println("2. Company");
                System.out.println("3. View Cache");
                System.out.println("4. Exit");
                System.out.print("Enter your choice: ");
                int choice = sc.nextInt();
                sc.nextLine(); // Consume newline

                // Handle user choice
                switch (choice) {
                    case 1:
                        handleEmployee();
                        break;
                    case 2:
                        handleCompany();
                        break;
                    case 3:
                        viewCache();
                        break;
                    case 4:
                        exit = true;
                        System.out.println("Exiting the portal.");
                        break;
                    default:
                        System.out.println("Invalid choice. Try again.");
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            // Close all database resources
            closeResources();
        }
    }
```

```java
// Method to preload any initial data
    public static void preloadData() {
        System.out.println("Pre-loaded data initialized successfully!");
    }

    // Method to handle employee-related operations
    public static void handleEmployee() {
        System.out.println("\n-- Employee Menu --");
        System.out.println("1. Enter New Details");
        System.out.println("2. Show Eligible Companies");
        System.out.print("Enter your choice: ");
        int choice = sc.nextInt();
        sc.nextLine();

        switch (choice) {
            case 1:
                addEmployee();
                break;
            case 2:
                showEligibleCompanies();
                break;
            default:
                System.out.println("Invalid choice.");
        }
    }

    // Method to handle company-related operations
    public static void handleCompany() {
        System.out.println("\n-- Company Menu --");
        System.out.println("1. Add More Details");
        System.out.println("2. Select an Employee");
        System.out.print("Enter your choice: ");
        int choice = sc.nextInt();
        sc.nextLine();

        switch (choice) {
            case 1:
                addCompany();
                break;
            case 2:
                showPreferredEmployees();
                break;
            default:
                System.out.println("Invalid choice.");
        }
    }

    // Method to add a new employee to the database
    public static void addEmployee() {
        System.out.print("Enter your name: ");
        String name = sc.nextLine();
        System.out.print("Enter your skills (comma separated, e.g., Java,Python): ");
        String skills = sc.nextLine();
        System.out.print("Enter your work experience (in years): ");
        int workExperience = sc.nextInt();

        String query = "INSERT INTO employees (first_name, skills, work_experience) VALUES (?, ?, ?)";
        try {
            preparedStatement = connection.prepareStatement(query);
            preparedStatement.setString(1, name);
            preparedStatement.setString(2, skills);
            preparedStatement.setInt(3, workExperience);
            preparedStatement.executeUpdate();
            System.out.println("Employee details added successfully!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Method to add a new company to the database
    public static void addCompany() {
        System.out.print("Enter company name: ");
        String name = sc.nextLine();
        System.out.print("Enter required skill (e.g., Java, Python): ");
        String skill = sc.nextLine();
        System.out.print("Enter minimum work experience required (in years): ");
        int minExperience = sc.nextInt();

        String query = "INSERT INTO companies (name, required_skill, min_experience) VALUES (?, ?, ?)";
        try {
            preparedStatement = connection.prepareStatement(query);
            preparedStatement.setString(1, name);
            preparedStatement.setString(2, skill);
            preparedStatement.setInt(3, minExperience);
            preparedStatement.executeUpdate();
            System.out.println("Company details added successfully!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

```java
// Method to show eligible companies for a given employee
    public static void showEligibleCompanies() {
        System.out.print("Enter your name to find eligible companies: ");
        String name = sc.nextLine();

        String query = "SELECT skills, work_experience FROM employees WHERE first_name = ?";
        try {
            preparedStatement = connection.prepareStatement(query);
            preparedStatement.setString(1, name);
            resultSet = preparedStatement.executeQuery();

            if (!resultSet.next()) {
                System.out.println("User not found. Please enter your details first.");
                return;
            }

            String skills = resultSet.getString("skills");
            int workExperience = resultSet.getInt("work_experience");
            String[] userSkills = skills.split(",");

            System.out.println("\nEligible Companies for " + name + ":");

            String companyQuery = "SELECT name FROM companies WHERE required_skill = ? AND min_experience <= ?";
            for (String skill : userSkills) {
                preparedStatement = connection.prepareStatement(companyQuery);
                preparedStatement.setString(1, skill.trim());
                preparedStatement.setInt(2, workExperience);
                resultSet = preparedStatement.executeQuery();

                while (resultSet.next()) {
                    System.out.println("Company: " + resultSet.getString("name"));
                }
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Method to show preferred employees for a given company
    public static void showPreferredEmployees() {
        System.out.print("Enter company name: ");
        String companyName = sc.nextLine();

        String query = "SELECT required_skill, min_experience FROM companies WHERE name = ?";
        try {
            preparedStatement = connection.prepareStatement(query);
            preparedStatement.setString(1, companyName);
            resultSet = preparedStatement.executeQuery();

            if (!resultSet.next()) {
                System.out.println("Company not found. Please enter company details first.");
                return;
            }

            String requiredSkill = resultSet.getString("required_skill");
            int minExperience = resultSet.getInt("min_experience");

            System.out.println("\nEmployees matching " + companyName + " requirements:");

            String employeeQuery = "SELECT first_name, skills, work_experience FROM employees WHERE skills LIKE ? AND work_experience >= ?";
            preparedStatement = connection.prepareStatement(employeeQuery);
            preparedStatement.setString(1, "%" + requiredSkill + "%");
            preparedStatement.setInt(2, minExperience);
            resultSet = preparedStatement.executeQuery();

            while (resultSet.next()) {
                System.out.println("Employee: " + resultSet.getString("first_name") +
                        " | Skill: " + resultSet.getString("skills") +
                        " | Work Experience: " + resultSet.getInt("work_experience") + " years");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Method to view the cached employees for frequently required skills
    public static void viewCache() {
        System.out.println("\n--- Cached Employees for Most Frequently Required Skills ---");
        for (Map.Entry<String, ArrayList<User>> entry : employeeCache.entrySet()) {
            String skill = entry.getKey();
            System.out.println("Skill: " + skill + " (Required by " + skillFrequency.getOrDefault(skill, 0) + " companies)");
            for (User user : entry.getValue()) {
                System.out.println("  Employee: " + user.name + " | Work Experience: " + user.workExperience + " years");
            }
        }
    }

    // Method to close all database resources
    public static void closeResources() {
        try {
            if (resultSet != null) {
                resultSet.close();
            }
            if (preparedStatement != null) {
                preparedStatement.close();
            }
            if (connection != null) {
                connection.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

# DAY 11 (30 September)



```java
// CODE 36
package com.example;

import java.sql.Connection;
import java.util.Map;
import com.google.common.cache.Cache;
import java.sql.Statement;
import java.sql.SQLException;
import java.sql.DriverManager;
import java.util.concurrent.TimeUnit;
import java.util.LinkedHashMap;
import com.google.common.cache.CacheBuilder;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.logging.Logger;
import java.util.logging.Level;
import java.util.logging.FileHandler;
import java.util.logging.SimpleFormatter;
import java.util.ArrayList;
import java.util.List;

public class AdvancedDatabaseCachingBenchmark {
    private static final String DB_URL = "jdbc:mysql://localhost:3306/testdb";
    private static final String DB_USER = "root";
    private static final String DB_PASSWORD = "root";
    private static final int NUM_ELEMENTS = 100000;
    private static final int L1_CACHE_SIZE = 10000;
    private static final int L2_CACHE_SIZE = 100000;
    private static final int L2_CACHE_DURATION_MINUTES = 10;
    private static final int PROGRESS_INTERVAL = 10000; // Log progress every 10,000 operations

    private static Connection connection;
    private static Map<Integer, String> l1Cache;
    private static Cache<Integer, String> l2Cache;
    private static final Logger LOGGER = Logger.getLogger(AdvancedDatabaseCachingBenchmark.class.getName());

    public static void main(String[] args) {
        setupLogging();
        try {
            setupDatabase();
            setupCaches();

            List<BenchmarkResult> results = new ArrayList<>();
            results.add(new BenchmarkResult("Database Insert", benchmarkDatabaseInsert()));
            results.add(new BenchmarkResult("Database Retrieve", benchmarkDatabaseRetrieve()));
            results.add(new BenchmarkResult("L1 Cache Insert", benchmarkL1CacheInsert()));
            results.add(new BenchmarkResult("L1 Cache Retrieve", benchmarkL1CacheRetrieve()));
            results.add(new BenchmarkResult("L2 Cache Insert", benchmarkL2CacheInsert()));
            results.add(new BenchmarkResult("L2 Cache Retrieve", benchmarkL2CacheRetrieve()));
            results.add(new BenchmarkResult("Multilevel Cache Retrieve", benchmarkMultilevelCacheRetrieve()));

            analyzeAndLogResults(results);
            prepareGraphData(results);

        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "An error occurred during benchmark execution", e);
        } finally {
            try {
                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException e) {
                LOGGER.log(Level.SEVERE, "Error closing database connection", e);
            }
        }
    }
```

```java
private static void setupLogging() {
        try {
            FileHandler fileHandler = new FileHandler("benchmark_log.txt");
            SimpleFormatter formatter = new SimpleFormatter();
            fileHandler.setFormatter(formatter);
            LOGGER.addHandler(fileHandler);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Error setting up logging", e);
        }
    }

    private static void setupDatabase() {
        try {
            connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
            Statement statement = connection.createStatement();
            statement.executeUpdate("CREATE TABLE IF NOT EXISTS test_table (id INT PRIMARY KEY, value VARCHAR(255))");
            statement.close();
            LOGGER.info("Database setup completed successfully.");
        } catch (SQLException e) {
            LOGGER.log(Level.SEVERE, "Error setting up database", e);
        }
    }

    private static void setupCaches() {
        l1Cache = new LinkedHashMap<Integer, String>(L1_CACHE_SIZE, 0.75f, true) {
            @Override
            protected boolean removeEldestEntry(Map.Entry<Integer, String> eldest) {
                return size() > L1_CACHE_SIZE;
            }
        };
        l2Cache = CacheBuilder.newBuilder()
                .maximumSize(L2_CACHE_SIZE)
                .expireAfterAccess(L2_CACHE_DURATION_MINUTES, TimeUnit.MINUTES)
                .build();
        LOGGER.info("Caches setup completed successfully.");
    }

    private static long benchmarkDatabaseInsert() throws SQLException {
        LOGGER.info("Starting Database Insert benchmark...");
        long startTime = System.nanoTime();
        String sql = "INSERT INTO test_table (id, value) VALUES (?, ?)";
        PreparedStatement statement = connection.prepareStatement(sql);
        for (int i = 0; i < NUM_ELEMENTS; i++) {
            statement.setInt(1, i);
            statement.setString(2, "Value" + i);
            statement.addBatch();
            if (i % 100 == 0) {
                statement.executeBatch();
            }
            logProgress("Database Insert", i);
        }
        statement.executeBatch();
        statement.close();
        long endTime = System.nanoTime();
        LOGGER.info("Database Insert benchmark completed.");
        return endTime - startTime;
    }

    private static long benchmarkDatabaseRetrieve() throws SQLException {
        LOGGER.info("Starting Database Retrieve benchmark...");
        long startTime = System.nanoTime();
        String sql = "SELECT * FROM test_table WHERE id = ?";
        PreparedStatement statement = connection.prepareStatement(sql);
        for (int i = 0; i < NUM_ELEMENTS; i++) {
            statement.setInt(1, i);
            ResultSet resultSet = statement.executeQuery();
            if (resultSet.next()) {
                resultSet.getString("value");
            }
            resultSet.close();
            logProgress("Database Retrieve", i);
        }
        statement.close();
        long endTime = System.nanoTime();
        LOGGER.info("Database Retrieve benchmark completed.");
        return endTime - startTime;
    }
```

```java
private static long benchmarkL1CacheInsert() {
        LOGGER.info("Starting L1 Cache Insert benchmark...");
        long startTime = System.nanoTime();
        for (int i = 0; i < NUM_ELEMENTS; i++) {
            l1Cache.put(i, "Value" + i);
            logProgress("L1 Cache Insert", i);
        }
        long endTime = System.nanoTime();
        LOGGER.info("L1 Cache Insert benchmark completed.");
        return endTime - startTime;
    }

    private static long benchmarkL1CacheRetrieve() {
        LOGGER.info("Starting L1 Cache Retrieve benchmark...");
        long startTime = System.nanoTime();
        for (int i = 0; i < NUM_ELEMENTS; i++) {
            l1Cache.get(i);
            logProgress("L1 Cache Retrieve", i);
        }
        long endTime = System.nanoTime();
        LOGGER.info("L1 Cache Retrieve benchmark completed.");
        return endTime - startTime;
    }

    private static long benchmarkL2CacheInsert() {
        LOGGER.info("Starting L2 Cache Insert benchmark...");
        long startTime = System.nanoTime();
        for (int i = 0; i < NUM_ELEMENTS; i++) {
            l2Cache.put(i, "Value" + i);
            logProgress("L2 Cache Insert", i);
        }
        long endTime = System.nanoTime();
        LOGGER.info("L2 Cache Insert benchmark completed.");
        return endTime - startTime;
    }

    private static long benchmarkL2CacheRetrieve() {
        LOGGER.info("Starting L2 Cache Retrieve benchmark...");
        long startTime = System.nanoTime();
        for (int i = 0; i < NUM_ELEMENTS; i++) {
            l2Cache.getIfPresent(i);
            logProgress("L2 Cache Retrieve", i);
        }
        long endTime = System.nanoTime();
        LOGGER.info("L2 Cache Retrieve benchmark completed.");
        return endTime - startTime;
    }

    private static long benchmarkMultilevelCacheRetrieve() throws SQLException {
        LOGGER.info("Starting Multilevel Cache Retrieve benchmark...");
        long startTime = System.nanoTime();
        for (int i = 0; i < NUM_ELEMENTS; i++) {
            String value = l1Cache.get(i);
            if (value == null) {
                value = l2Cache.getIfPresent(i);
                if (value == null) {
                    String sql = "SELECT * FROM test_table WHERE id = ?";
                    PreparedStatement statement = connection.prepareStatement(sql);
                    statement.setInt(1, i);
                    ResultSet resultSet = statement.executeQuery();
                    if (resultSet.next()) {
                        value = resultSet.getString("value");
                        l2Cache.put(i, value);
                        l1Cache.put(i, value);
                    }
                    resultSet.close();
                    statement.close();
                }
                l1Cache.put(i, value);
            } else {
                l2Cache.put(i, value);
                l1Cache.put(i, value);
            }
            logProgress("Multilevel Cache Retrieve", i);
        }
        long endTime = System.nanoTime();
        LOGGER.info("Multilevel Cache Retrieve benchmark completed.");
        return endTime - startTime;
    }
```

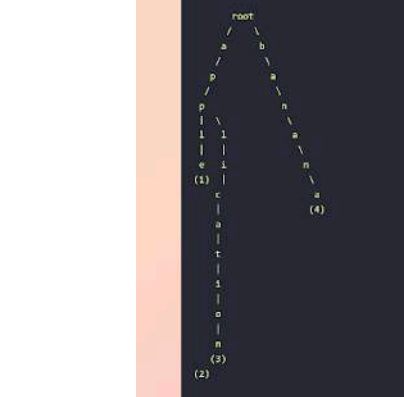
```java
private static void logProgress(String operation, int currentElement) {
        if (currentElement % PROGRESS_INTERVAL == 0) {
            LOGGER.info(operation + " progress: " + currentElement + " / " + NUM_ELEMENTS);
        }
    }

    private static void analyzeAndLogResults(List<BenchmarkResult> results) {
        LOGGER.info("Performance Analysis:");
        for (BenchmarkResult result : results) {
            logPerformance(result.operation, result.time);
        }

        LOGGER.info("\nPerformance Improvements:");
        BenchmarkResult dbRetrieve = results.stream()
                .filter(r -> r.operation.equals("Database Retrieve"))
                .findFirst()
                .orElseThrow();

        results.stream()
                .filter(r -> !r.operation.equals("Database Retrieve") && !r.operation.equals("Database Insert"))
                .forEach(r -> logImprovement(r.operation + " vs Database Retrieve", dbRetrieve.time, r.time));
    }

    private static void logPerformance(String operation, long timeNanos) {
        double timeMinutes = timeNanos / (60.0 * 1e9);
        LOGGER.info(String.format("%s: %.4f minutes", operation, timeMinutes));
    }

    private static void logImprovement(String comparison, long baseTime, long improvedTime) {
        double improvement = (baseTime - improvedTime) / (double) baseTime * 100;
        LOGGER.info(String.format("%s: %.2f%% faster", comparison, improvement));
    }

    private static void prepareGraphData(List<BenchmarkResult> results) {
        StringBuilder graphData = new StringBuilder("Operation,Time (minutes)\n");
        for (BenchmarkResult result : results) {
            double timeMinutes = result.time / (60.0 * 1e9);
            graphData.append(String.format("%s,%.4f\n", result.operation, timeMinutes));
        }
        LOGGER.info("Graph Data:\n" + graphData.toString());
    }

    private static class BenchmarkResult {
        String operation;
        long time;

        BenchmarkResult(String operation, long time) {
            this.operation = operation;
            this.time = time;
        }
    }
}
```

```java
// CODE 37
import java.util.Map;
import java.util.*;
public class AdvancedDataStructure {
    /*
     * Why trie map?
     * TrieMap is a data structure that is used to store a dynamic set of strings.
     * It is a tree-like structure that is used to store strings in a way that allows for efficient prefix search.
     * It is also known as a prefix tree.
     * It is used in many applications such as spell checking, autocomplete, and routing.
     * It is also used in many databases such as Redis.
     * It is also used in many search engines such as Elasticsearch.
     * It is also used in many routing algorithms such as Dijkstra's algorithm.
     * It is also used in many compression algorithms such as Huffman coding.
     * It is also used in many encryption algorithms such as AES.
     */
    private static void demostrateTrieMap(){
        System.out.println("Demostrating TrieMap");
        TrieMap<Integer> trieMap = new TrieMap<>();
        trieMap.put("apple", 1);
        trieMap.put("banana", 2);
        trieMap.put("orange", 3);
       // System.out.println("TrieMap size: " + trieMap.size());
        System.out.println("TrieMap contains key 'banana': " + trieMap.get("apple"));
        System.out.println("TrieMap contains key 'grape': " + trieMap.get("grape"));
        //System.out.println("TrieMap prefix search for 'app': " + trieMap.prefixSearch("app"));
        //System.out.println("TrieMap prefix search for 'ora': " + trieMap.prefixSearch("ora"));
        //System.out.println("TrieMap keys: " + trieMap.keys());
        //System.out.println("TrieMap values: " + trieMap.values());
    }
    public static void main(String[] args) {
    demostrateTrieMap();
    }

}
class TrieNode<T> {
    Map<Character, TrieNode<T>> children = new HashMap<>();
    T value;
    boolean isEndOfWord;
}
class TrieMap<T> {
    private TrieNode<T> root =new TrieNode<>();

    public void put(String key, T value){
        TrieNode<T> current = root;
        for(char c : key.toCharArray()){
            current.children.computeIfAbsent(c, t -> new TrieNode<>());
        }
        current.value = value;
        current.isEndOfWord = true;
    }
    public T get(String key){
        TrieNode<T> current = findNode(key);
        return current != null && current.isEndOfWord ? current.value : null;
    }
    private TrieNode<T> findNode(String key){
        TrieNode<T> current = root;
        for(char c : key.toCharArray()){
            current = current.children.get(c);
            if(current == null){
                return null;
            }
        }
        return current;
    }
    public boolean containsKey(String key){
        return get(key) != null;
    }
    public boolean hasPrefix(String prefix){
        return findNode(prefix) != null;
    }
    public List<String> getWordsForPrefix(String prefix){
        List<String> results = new ArrayList<>();
        TrieNode<T> node = findNode(prefix);
        if(node != null){
            collectAllWords(node, prefix, results);
        }
        return results;
    }
    private void collectAllWords(TrieNode<T> node, String prefix, List<String> results){
        if(node.isEndOfWord){
            results.add(prefix);
        }
        for(Map.Entry<Character, TrieNode<T>> entry : node.children.entrySet()){
            collectAllWords(entry.getValue(), prefix + entry.getKey(), results);
        }
    }
}
```

```java
// CODE 38

import java.util.*;

// TrieNode class represents a single node in the Trie
class TrieNode<T> {
    // Map to store child nodes, where the key is the character and value is the child TrieNode
    Map<Character, TrieNode<T>> children;

    // Value associated with this node (null if this node doesn't represent the end of a word)
    T value;

    // Flag to indicate if this node represents the end of a word
    boolean isEndOfWord;

    // Constructor to initialize a new TrieNode
    public TrieNode() {
        this.children = new HashMap<>();
        this.value = null;
        this.isEndOfWord = false;
    }
}

// Trie class implements the main functionality of the Trie data structure
class Trie<T> {
    // Root node of the Trie
    private TrieNode<T> root;

    // Constructor to initialize an empty Trie
    public Trie() {
        this.root = new TrieNode<>();
    }

    // Method to insert a word and its associated value into the Trie
    public void insert(String word, T value) {
        TrieNode<T> current = root;

        // Iterate through each character in the word
        for (char c : word.toCharArray()) {
            // If the current character doesn't exist as a child, create a new node
            current = current.children.computeIfAbsent(c, k -> new TrieNode<>());
        }

        // Mark the last node as the end of a word and set its value
        current.isEndOfWord = true;
        current.value = value;
    }

    // Method to search for a word in the Trie
    public T search(String word) {
        TrieNode<T> node = findNode(word);
        // Return the value if the word exists and is marked as end of word, otherwise null
        return (node != null && node.isEndOfWord) ? node.value : null;
    }

    // Method to check if any word in the Trie starts with the given prefix
    public boolean startsWith(String prefix) {
        // If we can find a node for this prefix, it exists in the Trie
        return findNode(prefix) != null;
    }
```

```java
// Helper method to find a node corresponding to a given word or prefix
    private TrieNode<T> findNode(String str) {
        TrieNode<T> current = root;

        // Traverse the Trie following the characters in the string
        for (char c : str.toCharArray()) {
            current = current.children.get(c);
            // If at any point we can't find a child node, the word/prefix doesn't exist
            if (current == null) return null;
        }

        // Return the final node we reached
        return current;
    }

    // Method to get all words in the Trie with a given prefix
    public List<String> getWordsWithPrefix(String prefix) {
        List<String> result = new ArrayList<>();
        TrieNode<T> prefixNode = findNode(prefix);

        // If the prefix exists in the Trie, collect all words starting from its last node
        if (prefixNode != null) {
            collectWords(prefixNode, prefix, result);
        }

        return result;
    }

    // Recursive helper method to collect all words from a given node
    private void collectWords(TrieNode<T> node, String currentPrefix, List<String> result) {
        // If this node is marked as end of word, add the current prefix to results
        if (node.isEndOfWord) {
            result.add(currentPrefix);
        }

        // Recursively explore all child nodes
        for (Map.Entry<Character, TrieNode<T>> entry : node.children.entrySet()) {
            collectWords(entry.getValue(), currentPrefix + entry.getKey(), result);
        }
    }
}

// Main class to demonstrate the usage of the Trie
public class TrieDemo {
    public static void main(String[] args) {
        Trie<Integer> trie = new Trie<>();

        // Insert some words with associated values
        trie.insert("apple", 1);
        trie.insert("app", 2);
        trie.insert("application", 3);
        trie.insert("banana", 4);

        // Demonstrate search functionality
        System.out.println("Value for 'apple': " + trie.search("apple"));  // Output: 1
        System.out.println("Value for 'app': " + trie.search("app"));      // Output: 2
        System.out.println("Value for 'appl': " + trie.search("appl"));    // Output: null

        // Demonstrate prefix checking
        System.out.println("Starts with 'app': " + trie.startsWith("app"));  // Output: true
        System.out.println("Starts with 'ban': " + trie.startsWith("ban"));  // Output: true
        System.out.println("Starts with 'car': " + trie.startsWith("car"));  // Output: false

        // Demonstrate getting words with a prefix
        System.out.println("Words with prefix 'app': " + trie.getWordsWithPrefix("app"));
        // Output: [app, apple, application]
    }
}
```

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




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 + "│   ");
    }
}
```

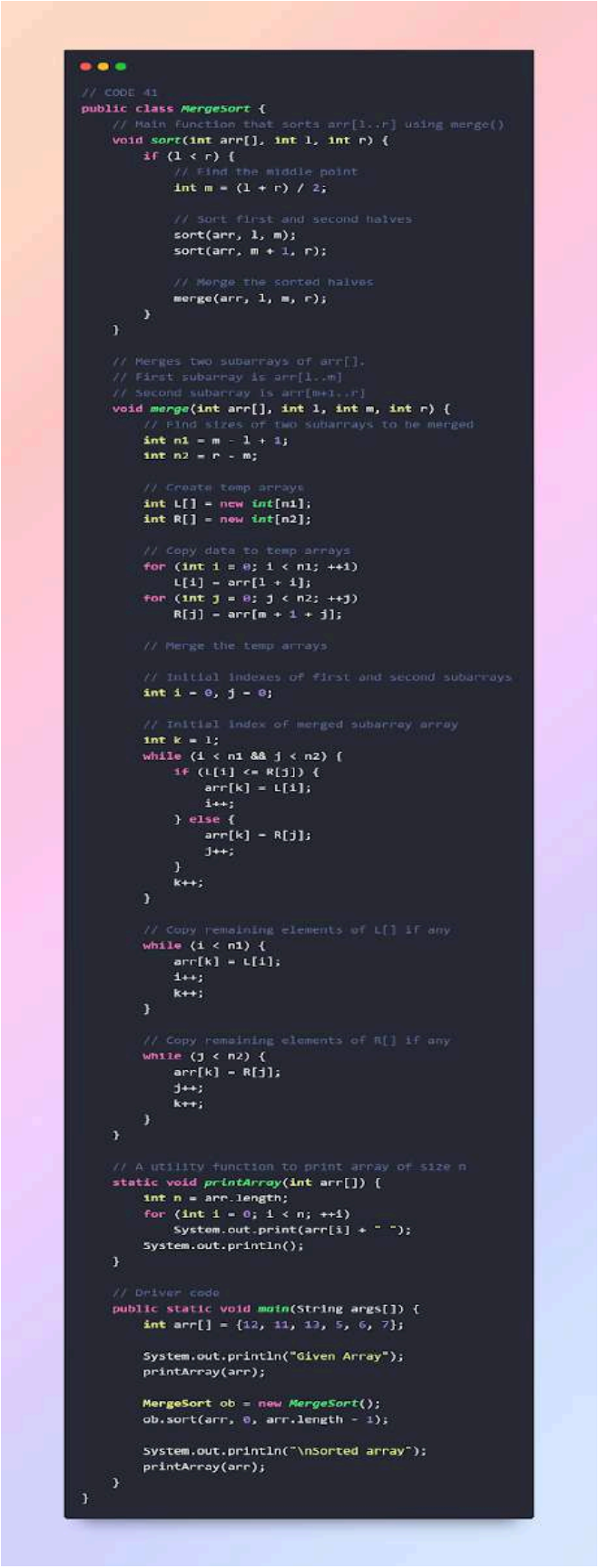
This would produce a visual representation of the Trie structure, helping in debugging and understanding the current state of the Trie.

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.

```java
// CODE 39

import java.util.*;

public class BinarySearch {

    public static int binarySearch(int[] nums, int target) {
        int n = nums.length; //size of the array.
        int low = 0, high = n - 1;

        // Perform the steps:
        while (low <= high) {
            int mid = (low + high) / 2;
            if (nums[mid] == target) return mid;
            else if (target > nums[mid]) low = mid + 1;
            else high = mid - 1;
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] a = {3, 4, 6, 7, 9, 12, 16, 17};
        int target = 6;
        int ind = binarySearch(a, target);
        if (ind == -1)
            System.out.println("The target is not present.");
        else
            System.out.println("The target is at index: " + ind);
    }
}
```

```java
// CODE 40
int binarySearch(int arr[], int low, int high, int x)
    {
        if (high >= low) {
            int mid = low + (high - low) / 2;

            // If the element is present at the
            // middle itself
            if (arr[mid] == x)
                return mid;

            // If element is smaller than mid, then
            // it can only be present in left subarray
            if (arr[mid] > x)
                return binarySearch(arr, low, mid - 1, x);

            // Else the element can only be present
            // in right subarray
            return binarySearch(arr, mid + 1, high, x);
        }

        // We reach here when element is not present
        // in array
        return -1;
    }
```

# DAY 12 (1 October)



```java
// CODE 41
public class MergeSort {
    // Main function that sorts arr[l..r] using merge()
    void sort(int arr[], int l, int r) {
        if (l < r) {
            // Find the middle point
            int m = (l + r) / 2;

            // Sort first and second halves
            sort(arr, l, m);
            sort(arr, m + 1, r);

            // Merge the sorted halves
            merge(arr, l, m, r);
        }
    }

    // Merges two subarrays of arr[].
    // First subarray is arr[l..m]
    // Second subarray is arr[m+1..r]
    void merge(int arr[], int l, int m, int r) {
        // Find sizes of two subarrays to be merged
        int n1 = m - l + 1;
        int n2 = r - m;

        // Create temp arrays
        int L[] = new int[n1];
        int R[] = new int[n2];

        // Copy data to temp arrays
        for (int i = 0; i < n1; ++i)
            L[i] = arr[l + i];
        for (int j = 0; j < n2; ++j)
            R[j] = arr[m + 1 + j];

        // Merge the temp arrays

        // Initial indexes of first and second subarrays
        int i = 0, j = 0;

        // Initial index of merged subarray array
        int k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k] = L[i];
                i++;
            } else {
                arr[k] = R[j];
                j++;
            }
            k++;
        }

        // Copy remaining elements of L[] if any
        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }

        // Copy remaining elements of R[] if any
        while (j < n2) {
            arr[k] = R[j];
            j++;
            k++;
        }
    }

    // A utility function to print array of size n
    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver code
    public static void main(String args[]) {
        int arr[] = {12, 11, 13, 5, 6, 7};

        System.out.println("Given Array");
        printArray(arr);

        MergeSort ob = new MergeSort();
        ob.sort(arr, 0, arr.length - 1);

        System.out.println("\nSorted array");
        printArray(arr);
    }
}
```

```java
// CODE 42
public class QuickSort {

    // This function takes last element as pivot,
    // places the pivot element at its correct position in sorted array,
    // and places all smaller elements to left of pivot and
    // all greater elements to right of pivot
    int partition(int arr[], int low, int high) {
        int pivot = arr[high];
        int i = (low - 1); // index of smaller element
        for (int j = low; j < high; j++) {
            // If current element is smaller than or equal to pivot
            if (arr[j] <= pivot) {
                i++;

                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }

    // The main function that implements QuickSort
    // arr[] --> Array to be sorted,
    // low --> Starting index,
    // high --> Ending index
    void sort(int arr[], int low, int high) {
        if (low < high) {
            // pi is partitioning index, arr[pi] is now at right place
            int pi = partition(arr, low, high);

            // Recursively sort elements before partition and after partition
            sort(arr, low, pi - 1);
            sort(arr, pi + 1, high);
        }
    }

    // A utility function to print array of size n
    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver program
    public static void main(String args[]) {
        int arr[] = {10, 7, 8, 9, 1, 5};
        int n = arr.length;

        QuickSort ob = new QuickSort();
        ob.sort(arr, 0, n - 1);

        System.out.println("Sorted array:");
        printArray(arr);
    }
}
```

```java
// CODE 43
class Node {
    int key;
    Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}
public class BST {
    // A utility function to insert a new node
    // with the given key
    static Node insert(Node root, int key)
    {

        // If the tree is empty, return a new node
        if (root == null)
            return new Node(key);

        // If the key is already present in the tree,
        // return the node
        if (root.key == key)
            return root;

        // Otherwise, recur down the tree
        if (key < root.key)
            root.left = insert(root.left, key);
        else
            root.right = insert(root.right, key);

        // Return the (unchanged) node pointer
        return root;
    }

    // function to search a key in a BST
    static Node search(Node root, int key)
    {
        // Base Cases: root is null or key is present at
        // root
        if (root == null || root.key == key)
            return root;

        // Key is greater than root's key
        if (root.key < key)
            return search(root.right, key);

        // Key is smaller than root's key
        return search(root.left, key);
    }

    // A utility function to do inorder tree traversal
    static void inorder(Node root)
    {
        if (root != null) {
            inorder(root.left);
            System.out.print(root.key + " ");
            inorder(root.right);
        }
    }
```

```java
public static void preorder(Node node)
    {
        if (node == null)
        return;

        System.out.print(node.key + " ");
        preorder(node.left);
        preorder(node.right);
    }

    static void postorder(Node node)
    {
        if (node == null)
            return;

        postorder(node.left);
        postorder(node.right);
        System.out.print(node.key + " ");
    }
    // Driver method
    public static void main(String[] args)
    {
        Node root = null;

        // Creating the following BST
        //      50
        //     /  \
        //    30   70
        //   / \   / \
        //  20 40 60 80

        root = insert(root, 50);
        root = insert(root, 30);
        root = insert(root, 20);
        root = insert(root, 40);
        root = insert(root, 70);
        root = insert(root, 60);
        root = insert(root, 80);

        // Print inorder traversal of the BST
        inorder(root);
        System.out.println();
        preorder(root);
        System.out.println();
        postorder(root);
        Node result = search(root, 40);
        System.out.println("\n" + result.key);
    }
}
```

```java
// CODE 44
  Node deleteRec(Node root, int key) {
        // Base case: If the tree is empty
        if (root == null) return null;

        // Recursive calls for ancestors of node to be deleted
        if (key < root.data)
            root.left = deleteRec(root.left, key);
        else if (key > root.data)
            root.right = deleteRec(root.right, key);
        // We reach here when root is the node to be deleted.
        else {
            // Case 1: Node with only one child or no child
            if (root.left == null)
                return root.right;
            else if (root.right == null)
                return root.left;

            // Case 2: Node with two children
            // Find the inorder successor (smallest in the right subtree)
            Node successor = root.right;
            Node successorParent = root;

            while (successor.left != null) {
                successorParent = successor;
                successor = successor.left;
            }

            // Replace root's data with successor's data
            root.data = successor.data;

            // Delete the successor
            if (successorParent != root)
                successorParent.left = successor.right;
            else
                successorParent.right = successor.right;
        }
        return root;
    }
```

# Detailed Dry Run of BST Deletion

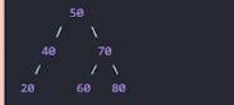
Let's start with a binary search tree containing 7 nodes:


## 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:

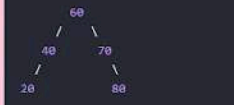



## Second Deletion: Node 50

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.

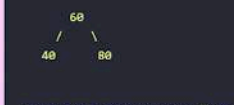
The tree now looks like this:


## 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.

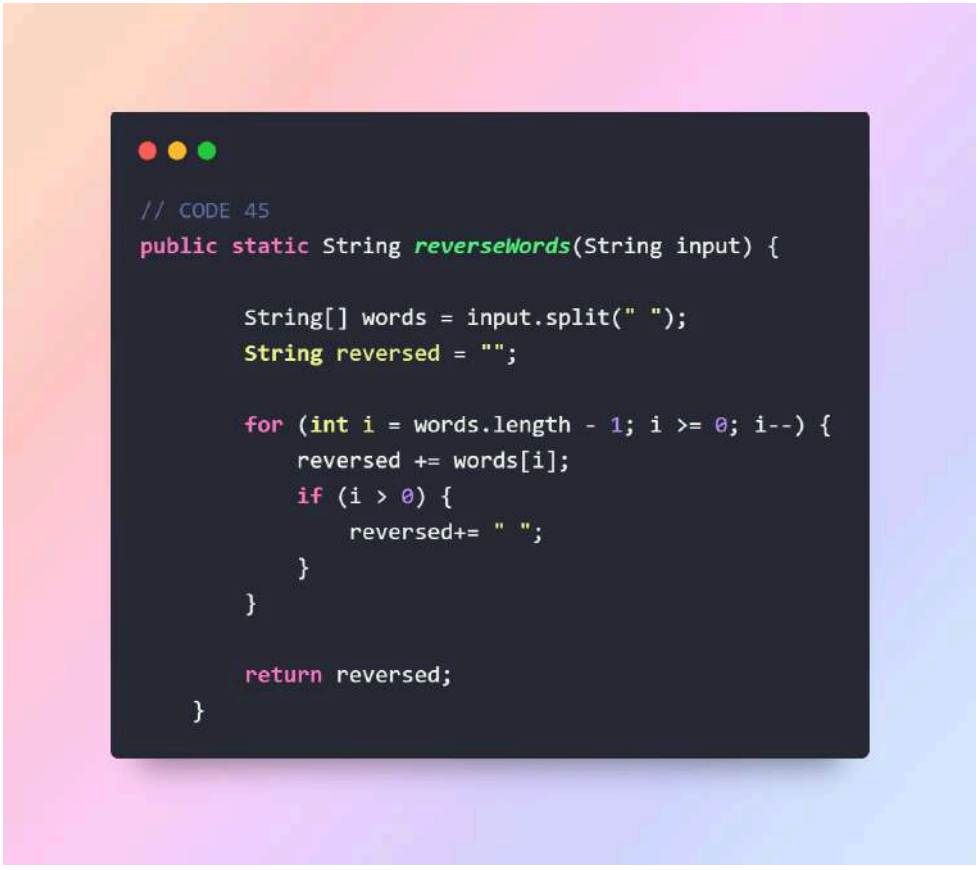
The final tree looks like this:




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)

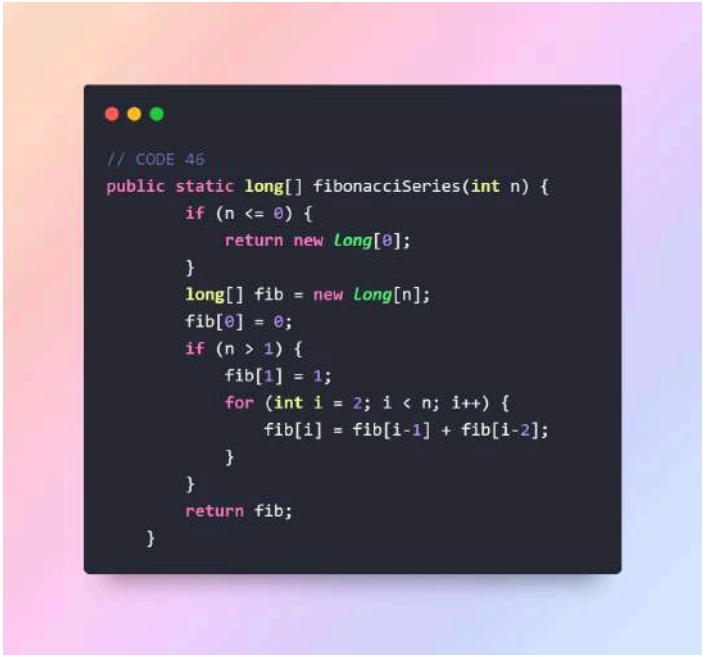
In each case, the binary search tree property is maintained, ensuring that for each node, all elements in its left subtree are smaller and all elements in its right subtree are larger.

# DAY 13 (3 October)



```java
// CODE 45
public static String reverseWords(String input) {

        String[] words = input.split(" ");
        String reversed = "";

        for (int i = words.length - 1; i >= 0; i--) {
            reversed += words[i];
            if (i > 0) {
                reversed+= " ";
            }
        }

        return reversed;
    }
```

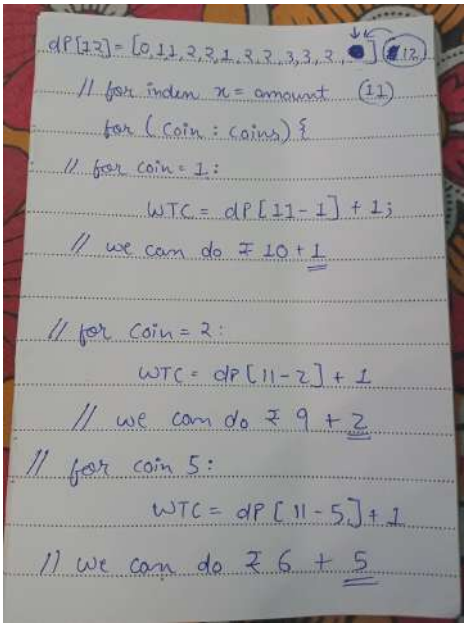
# DYNAMIC  PROGRAMMING:



```java
// CODE 46
public static long[] fibonacciSeries(int n) {
        if (n <= 0) {
            return new long[0];
        }
        long[] fib = new long[n];
        fib[0] = 0;
        if (n > 1) {
            fib[1] = 1;
            for (int i = 2; i < n; i++) {
                fib[i] = fib[i-1] + fib[i-2];
            }
        }
        return fib;
    }
```

```java
// CODE 47
public static int coinChange(int[] coins, int amount) {
        int[] dp = new int[amount + 1];
        Arrays.fill(dp, amount + 1);
        dp[0] = 0;

        for (int x = 1; x <= amount; x++) {
            for (int coin : coins) {
                if (coin <= x) {
                    int withThisCoin = dp[x - coin] + 1;

                    if (withThisCoin < dp[x]) {
                        dp[x] = withThisCoin;
                    }
                }
            }
            System.out.println("Best for amount " + x + ": " + dp[x] + " coins");
        }

        return dp[amount] > amount ? -1 : dp[amount];
    }
```

## DRY RUN:



```java
// CODE 48
public static boolean checkAnagram(String str1, String str2) {

        if(str1.length()!=str2.length()) {
            return false;
        }

        int arr1[] = new int[26];
        int arr2[] = new int[26];
        for(int i=0; i<str1.length(); i++){
            int index = str1.charAt(i)-'a';
            arr1[index]++;
        }
        for(int i=0; i<str2.length(); i++){
            int index = str2.charAt(i)-'a';
            arr2[index]++;
        }
        for(int i=0; i<26; i++){
            if(arr1[i]!=arr2[i])
                return false;
        }
        return true;  // If all characters have same frequency, then str1 and str2 are anagrams.
    }
```

# PRACTICE PROBLEM:

## 819. Most Common Word

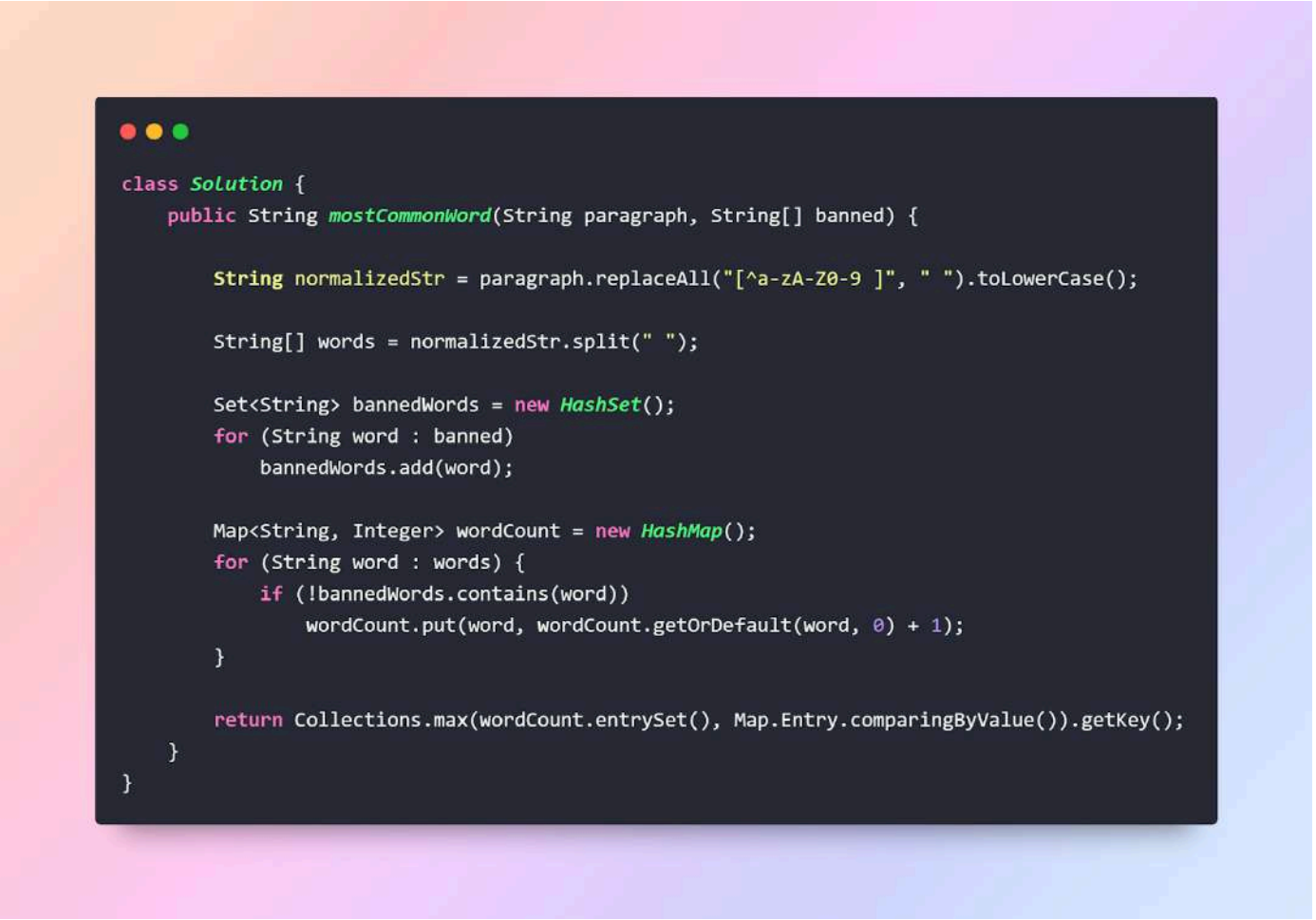
**Given a string paragraph and a string array of the banned words banned, return *the most frequent word that is not banned*. It is guaranteed there is at least one word that is not banned, and that the answer is unique.**

**The words in paragraph are case-insensitive and the answer should be returned in lowercase.**

**Example 1:**

**Input: paragraph = "Bob hit a ball, the hit BALL flew far after it was hit.", banned = ["hit"]**
**Output: "ball"**

# SOLUTION:



```java
class Solution {
    public String mostCommonWord(String paragraph, String[] banned) {

        String normalizedStr = paragraph.replaceAll("[^a-zA-Z0-9 ]", " ").toLowerCase();

        String[] words = normalizedStr.split(" ");

        Set<String> bannedWords = new HashSet();
        for (String word : banned)
            bannedWords.add(word);

        Map<String, Integer> wordCount = new HashMap();
        for (String word : words) {
            if (!bannedWords.contains(word))
                wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
        }

        return Collections.max(wordCount.entrySet(), Map.Entry.comparingByValue()).getKey();
    }
}
```

## DAY 14 (4 October)

Doubt asked by me : Difference between split(“ “) and split(“//s+”)

`split(" ")`: Splits the string based on single spaces only.
`split("\\s+")`: Splits the string based on one or more whitespace characters (spaces, tabs, line breaks).

```java
// CODE 49
public static String longestPalindrome(String s) {
        if (s == null || s.length() < 2) {
            return s;
        }

        int start = 0, maxLength = 1;

        for (int i = 0; i < s.length(); i++) {
            int len1 = expandAroundCenter(s, i, i);
            int len2 = expandAroundCenter(s, i, i + 1);
            int len = Math.max(len1, len2);

            if (len > maxLength) {
                start = i - (len - 1) / 2;
                maxLength = len;
            }
        }
        return s.substring(start, start + maxLength);
    }

    private static int expandAroundCenter(String s, int left, int right) {
        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
            left--;
            right++;
        }
        return right - left - 1;
    }
```

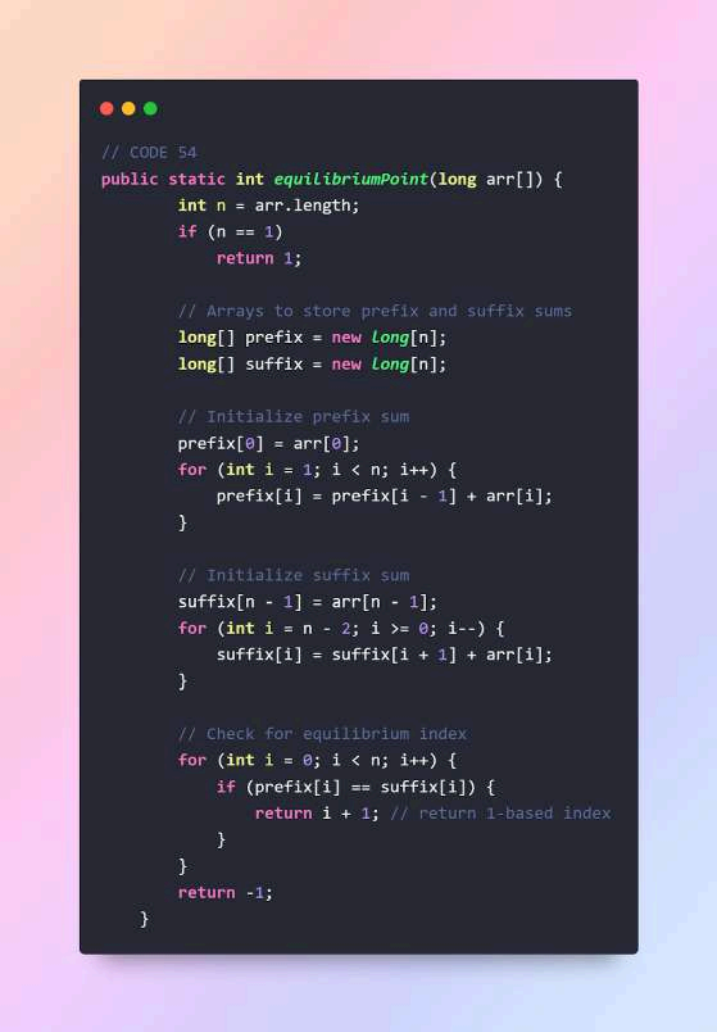
```java
// CODE 50
public static int[] removeDuplicates(int[] arr) {
        if (arr == null || arr.length == 0) {
            return new int[0];
        }
        // Count the frequency of each element
        HashMap<Integer, Integer> map = new HashMap<Integer,Integer>();
        for(int i = 0; i < arr.length; i++) {
            map.put(arr[i], map.getOrDefault(arr[i], 0) + 1);
        }

        // Create a list to store elements that appear only once
        List<Integer> uniqueElements = new ArrayList<>();
        for (Map.Entry<Integer, Integer> entry : map.entrySet()) {
            if (entry.getValue() == 1) {
                uniqueElements.add(entry.getKey());
            }
        }

        // Convert the list to an array
        int[] result = new int[uniqueElements.size()];
        for (int i = 0; i < uniqueElements.size(); i++) {
            result[i] = uniqueElements.get(i);
        }

        return result;
    }
```

```java
// CODE 51
package demo;

import java.security.KeyStore.Entry;

class HashTable<K,V>
{
    private Entry<K,V>[] table;
    private  int size;
    private static final double LOAD_FACTOR_THRESHOLD=0.75;

    public HashTable(int capacity){
        table =new Entry[capacity];
        size = 0;
    }
    public void put(K key, V value){
                if(key == null){ // Check if the key is null to prevent null pointer exceptions
            throw new IllegalArgumentException("Null keys are not allowed"); // Throw an exception if the key is null
        }
        if((double)size / table.length >= LOAD_FACTOR_THRESHOLD){ // Check if the load factor has exceeded the threshold
            resize(); // Resize the table if the load factor has exceeded the threshold
        }
        int index = hash1(key); // Calculate the initial index using the first hash function
        int step = hash2(key); // Calculate the step size using the second hash function
        int i = 0; // Initialize the iteration counter
        while(table[index] != null && !table[index].getKey().equals(key)){ // Loop until an empty slot is found or the key is found
            index = (index + step * i) % table.length; // Calculate the next index using the step size and iteration counter
            i++; // Increment the iteration counter
        }
        table[index] = new Entry<>(key, value); // Insert the new entry into the table
        size++; // Increment the size of the table

    }
    public V get(K key){
        if(key == null){ // Check if the key is null to prevent null pointer exceptions
            {
                throw new IllegalArgumentException("Null keys are not allowed"); // Throw an exception if the key is null
            }
        int index = hash1(key);
        int step = hash2(key);
        int i = 0;
        while(table[index]!=null){
            if(table[index].getKey.equals(key)){
                return table[index].getValue();
            }
            index = (index + step * i) % table.length;
            i++;
        }
        return null;
    }
    private void resize(){
        Entry<K,V> oldTable =table;
        table= new Entry[oldTable.length *2];
        size = 0;
        for (Entry<K,V> entry:oldTable){
            if (entry !=null){
                put(entry.getKey(),entry.getValue());
            }
        }
    }
    private int hash1(K key){
        return Math.abs(key.hashCode())%table.length;
    }
    private int hash2(K key){
        return 1+ (Math.abs(key.hashCode())%(table.length -1));
    }
    private static class Entry<K,V>{
        private K key;
        private V value;
        public Entry(K key, V value){
            this.key = key;
            this.value = value;
        }
        public K getKey(){
            return key;
        }
        public V getValue(){
            return value;
        }
    }
}
```

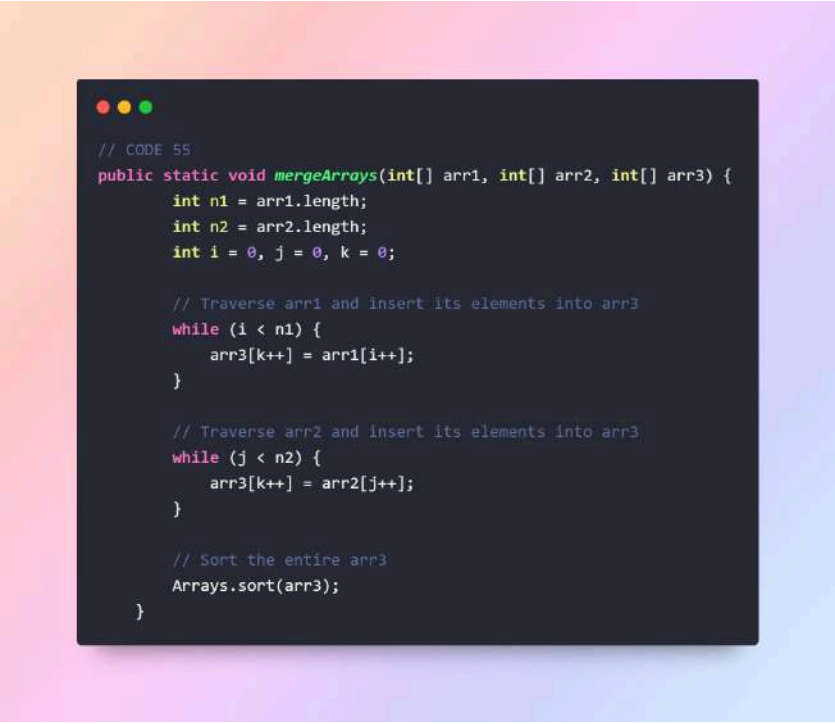
```java
// CODE 52
static ArrayList<Integer> leaders(int[] arr) {
        ArrayList<Integer> result = new ArrayList<>();
        int n = arr.length;

        // Start with the rightmost element
        int maxRight = arr[n - 1];

        // Rightmost element is always a leader
        result.add(maxRight);

        // Traverse the array from right to left
        for (int i = n - 2; i >= 0; i--) {
            if (arr[i] > maxRight) {
                maxRight = arr[i];
                result.add(maxRight);
            }
        }

        // Reverse the result list to maintain
        // original order
        Collections.reverse(result);

        return result;
    }
```

```java
// CODE 53
public static int[] searchMatrix(int[][] matrix, int target) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
            return new int[]{-1, -1};
        }

        int m = matrix.length;
        int n = matrix[0].length;
        int left = 0;
        int right = m * n - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;
            int midValue = matrix[mid / n][mid % n];

            if (midValue == target) {
                return new int[]{mid / n, mid % n};
            } else if (midValue < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        return new int[]{-1, -1};
    }
```

# PRACTICE PROBLEMS:



```java
// CODE 54
public static int equilibriumPoint(long arr[]) {
        int n = arr.length;
        if (n == 1)
            return 1;

        // Arrays to store prefix and suffix sums
        long[] prefix = new long[n];
        long[] suffix = new long[n];

        // Initialize prefix sum
        prefix[0] = arr[0];
        for (int i = 1; i < n; i++) {
            prefix[i] = prefix[i - 1] + arr[i];
        }

        // Initialize suffix sum
        suffix[n - 1] = arr[n - 1];
        for (int i = n - 2; i >= 0; i--) {
            suffix[i] = suffix[i + 1] + arr[i];
        }

        // Check for equilibrium index
        for (int i = 0; i < n; i++) {
            if (prefix[i] == suffix[i]) {
                return i + 1; // return 1-based index
            }
        }
        return -1;
    }
```



```java
// CODE 55
public static void mergeArrays(int[] arr1, int[] arr2, int[] arr3) {
        int n1 = arr1.length;
        int n2 = arr2.length;
        int i = 0, j = 0, k = 0;

        // Traverse arr1 and insert its elements into arr3
        while (i < n1) {
            arr3[k++] = arr1[i++];
        }

        // Traverse arr2 and insert its elements into arr3
        while (j < n2) {
            arr3[k++] = arr2[j++];
        }

        // Sort the entire arr3
        Arrays.sort(arr3);
    }
```

```java
// CODE 56
public static boolean ispar(String s) {
        // Declare a stack to hold the previous brackets.
        Stack<Character> stk = new Stack<>();
        for (int i = 0; i < s.length(); i++) {

            // Check if the character is an opening bracket
            if (s.charAt(i) == '(' || s.charAt(i) == '{' || s.charAt(i) == '[') {
                stk.push(s.charAt(i));
            }
            else {

                // If it's a closing bracket, check if the stack is non-empty
                // and if the top of the stack is a matching opening bracket
                if (!stk.empty() &&
                    ((stk.peek() == '(' && s.charAt(i) == ')') ||
                     (stk.peek() == '{' && s.charAt(i) == '}') ||
                     (stk.peek() == '[' && s.charAt(i) == ']'))) {
                    stk.pop();
                }
                else {
                    return false; // Unmatched closing bracket
                }
            }
        }

        // If stack is empty, return true (balanced),
        // otherwise false
        return stk.empty();
    }
```

THE ABOVE THREE PRCTICE PROBLEMS ARE:

1. Equilibrium point of array
2. Merge 2 sorted arrays
3. Check valid parenthesis

# DAY 15 (7 October)

## SQL (Structured Query Language):

- SQL is a standardized programming language used to manage and manipulate relational databases.
- It allows users to create, read, update, and delete (CRUD) data in databases.

### SQL Operations

Data Definition Language (DDL) – Defines database schema.

- `CREATE`, `ALTER`, `DROP`, etc.

Example:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT,
    Department VARCHAR(50)
);
```

Data Manipulation Language (DML) – Manages data within schema objects.

- `SELECT`, `INSERT`, `UPDATE`, `DELETE`.
- Example:
- Insert a new employee:
- `INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department)`
- `VALUES (1, 'John', 'Doe', 30, 'HR');`

- Update employee age:

- ```
  UPDATE Employees
  SET Age = 31
  WHERE EmployeeID = 1;
  ```

Data Query Language (DQL) – Queries data from the database.

- `SELECT`.

Example:

- Retrieve all employees:

  ```
  SELECT * FROM Employees;
  ```

Data Control Language (DCL) – Controls access to data.

- `GRANT`, `REVOKE`.

Example:

- Grant permissions to a user:

  ```
  GRANT SELECT ON Employees TO user_name;
  ```

Transaction Control Language (TCL) – Manages transactions within a database.

- `COMMIT`, `ROLLBACK`.

Example:

- Commit a transaction:

  ```
  COMMIT;
  ```
  ```

Query Examples

Basic SELECT Query:
Retrieve all employees who work in the 'HR' department.

```
SELECT FirstName, LastName, Age
FROM Employees
WHERE Department = 'HR';
```

1. JOIN Query:
   Retrieve employee information along with their department name from two tables: Employees and Departments.

```
SELECT Employees.FirstName, Employees.LastName,
Departments.DepartmentName
FROM Employees
JOIN Departments ON Employees.DepartmentID =
Departments.DepartmentID;
```

2. GROUP BY:
   Find the average age of employees in each department.

```
SELECT Department, AVG(Age) AS AverageAge
FROM Employees
GROUP BY Department;
```

3. HAVING:
   Get departments with an average employee age greater than 30.

```
SELECT Department, AVG(Age) AS AverageAge
FROM Employees
GROUP BY Department
HAVING AVG(Age) > 30;
```

4. ORDER BY:
   Retrieve all employees, ordered by their last name.

```
SELECT FirstName, LastName, Age

FROM Employees
ORDER BY LastName ASC;
```

5. Subquery:
   Find all employees who are older than the average age of employees.

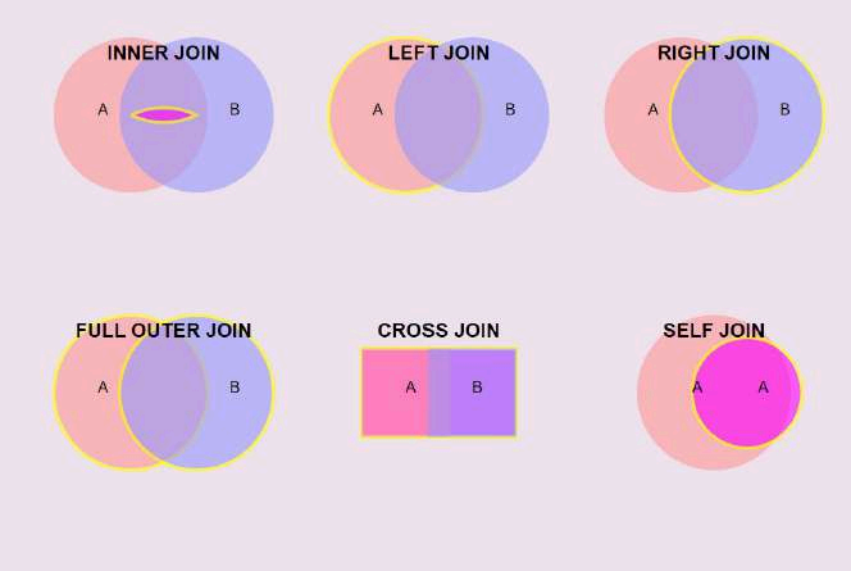
```
SELECT FirstName, LastName, Age

FROM Employees
WHERE Age > (SELECT AVG(Age) FROM Employees);
```

6. Conclusion

- SQL is crucial for database management, enabling efficient storage and retrieval of data.
- It supports a wide range of operations for handling relational data, including creating, querying, updating, and managing access to databases.

## CLASS WORK:



```sql
-- Create tables

CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50),
    location_id INT
);

CREATE TABLE locations (
    location_id INT PRIMARY KEY,
    city VARCHAR(50)
);

CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    phone_number VARCHAR(20),
    hire_date DATE,
    job_id VARCHAR(10),
    salary DECIMAL(10, 2),
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(100),
    email VARCHAR(100)
);

CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    price DECIMAL(10, 2)
);

CREATE TABLE discontinued_products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50)
);

-- Insert sample data
INSERT INTO departments (department_id, department_name, location_id) VALUES
(10, 'Administration', 1),
(20, 'Marketing', 2),
(30, 'Purchasing', 1),
(40, 'Human Resources', 2),
(50, 'Shipping', 3);

INSERT INTO locations (location_id, city) VALUES
(1, 'New York'),
(2, 'Los Angeles'),
(3, 'Chicago');

INSERT INTO employees (employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary, department_id) VALUES
(1, 'John', 'Doe', 'john.doe@example.com', '515.123.4567', '2019-06-17', 'AD_PRES', 24000.00, 10),
(2, 'Jane', 'Smith', 'jane.smith@example.com', '515.123.4568', '2020-02-20', 'AD_VP', 17000.00, 10),
(3, 'Alice', 'Johnson', 'alice.johnson@example.com', '515.123.4569', '2020-08-11', 'MK_MAN', 9000.00, 20),
(4, 'Bob', 'Brown', 'bob.brown@example.com', '515.123.4560', '2021-03-05', 'HR_REP', 6000.00, 40),
(5, 'Charlie', 'Davis', 'charlie.davis@example.com', '515.123.4561', '2021-11-30', 'SH_CLERK', 3000.00, 50);
```

```sql
INSERT INTO customers (customer_id, customer_name, email) VALUES
(1, 'Acme Corp', 'contact@acmecorp.com'),
(2, 'GlobalTech', 'info@globaltech.com'),
(3, 'Local Shop', 'owner@localshop.com');

INSERT INTO orders (order_id, customer_id, order_date) VALUES
(1, 1, '2023-01-15'),
(2, 1, '2023-02-20'),
(3, 2, '2023-02-22');

INSERT INTO products (product_id, product_name, category, price) VALUES
(1, 'Laptop', 'Electronics', 999.99),
(2, 'Smartphone', 'Electronics', 699.99),
(3, 'Desk Chair', 'Furniture', 199.99);

INSERT INTO discontinued_products (product_id, product_name, category) VALUES
(101, 'Old Laptop Model', 'Electronics'),
(102, 'Discontinued Phone', 'Electronics');

select * from employees;

select e.first_name, e.last_name, d.department_name
from employees e
inner join departments d
on e.department_id= d.department_id;


select c.customer_name , o.order_id
from customers c
left join orders o on c.customer_id =o.customer_id;



select c.customer_name , o.order_id
from customers c
right join orders o on c.customer_id =o.customer_id;


select first_name,last_name, salary
from employees
where salary >(select avg(salary) from employees);

select department_id , avg(salary) as avgsalary
from employees
group by department_id
having avgsalary >10000;

select c.customer_name,
o.order_date,
case
  when DAYOFWEEK(o.order_date) in (1,7) Then 'Weekend'
  else 'Weekday'
end as order_day_type,
p.product_name,
p.price,
case
  when p.price<100 then 'Budget'
  when p.price between 100 and 500 then 'mid range'
  else 'premium'
end as price_category
from customers c
inner join
orders o on  c.customer_id =o.customer_id
inner join products p on p.product_id in (select product_id from orders where order_id =o.order_id)
where
o.order_date <= DATE_SUB(CURDATE(),INTERVAL 1 YEAR)
order by
o.order_date desc;
```

```sql
-- Example 2: Product Recommendations
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    price DECIMAL(10, 2)
);

CREATE TABLE product_recommendations (
    product_id INT,
    recommended_product_id INT,
    strength DECIMAL(3, 2),
    PRIMARY KEY (product_id, recommended_product_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id),
    FOREIGN KEY (recommended_product_id) REFERENCES products(product_id)
);

INSERT INTO products (product_id, product_name, category, price) VALUES
(1, 'Laptop', 'Electronics', 999.99),
(2, 'Smartphone', 'Electronics', 699.99),
(3, 'Tablet', 'Electronics', 399.99),
(4, 'Headphones', 'Electronics', 149.99),
(5, 'Smart Watch', 'Electronics', 249.99);

INSE;RT INTO product_recommendations (product_id, recommended_product_id, strength) VALUES
(1, 2, 0.8),
(1, 3, 0.6),
(1, 4, 0.7),
(2, 1, 0.8),
(2, 3, 0.9),
(2, 5, 0.7),
(3, 1, 0.6),
(3, 2, 0.9),
(3, 4, 0.5)

-- Self join to get product recommendations with product names


select p1.product_name, p2.product_name, pr.strength
from product_recommendations pr join products p1
on p1.product_id = pr.product_id
join products p2 on pr.recommended_product_id = p2.product_id
where p1.product_id = 1
order by pr.strength desc;

select *
from product_recommendations pr join products p1
on p1.product_id = pr.product_id
join products p2 on pr.recommended_product_id = p2.product_id
order by pr.strength desc;
```

# DAY 16 (8 October)

```sql
-- Example 3: Flight Connections
CREATE TABLE airports (
    airport_code CHAR(3) PRIMARY KEY,
    airport_name VARCHAR(100),
    city VARCHAR(50),
    country VARCHAR(50)
);

CREATE TABLE flights (
    flight_id INT PRIMARY KEY,
    departure_airport CHAR(3),
    arrival_airport CHAR(3),
    departure_time TIME,
    arrival_time TIME,
    FOREIGN KEY (departure_airport) REFERENCES airports(airport_code),
    FOREIGN KEY (arrival_airport) REFERENCES airports(airport_code)
);

INSERT INTO airports (airport_code, airport_name, city, country) VALUES
('JFK', 'John F. Kennedy International Airport', 'New York', 'USA'),
('LAX', 'Los Angeles International Airport', 'Los Angeles', 'USA'),
('LHR', 'London Heathrow Airport', 'London', 'UK'),
('CDG', 'Charles de Gaulle Airport', 'Paris', 'France'),
('NRT', 'Narita International Airport', 'Tokyo', 'Japan');

INSERT INTO flights (flight_id, departure_airport, arrival_airport, departure_time, arrival_time) VALUES
(1, 'JFK', 'LAX', '08:00', '11:00'),
(2, 'LAX', 'NRT', '13:00', '17:00'),
(3, 'NRT', 'LHR', '19:00', '23:00'),
(4, 'LHR', 'CDG', '09:00', '10:30'),
(5, 'CDG', 'JFK', '12:00', '14:00'),
(6, 'JFK', 'LHR', '18:00', '06:00'),
(7, 'LHR', 'LAX', '10:00', '13:00');

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

select
f1.flight_id,
a1.city as departure_city,
a2.city as layover_city,
a3.city as arrival_city,
f1.departure_time,
f1.arrival_time as layover_arrival,
f2.departure_time as  layover_departure,
f2.arrival_time
from flights f1
join flights f2 on f1.arrival_airport =f2.departure_airport
join airports a1 on f1.departure_airport = a1.airport_code
join airports a2 on f1.arrival_airport = a2.airport_code
join airports a3 on f2.arrival_airport = a3.airport_code
where f1.departure_airport ='JFK'
and f2.arrival_airport ='NRT'
and f2.departure_time> f1.arrival_time;
```

```sql
CREATE database DB8;
USE DB8;

-- Sample Data Setup
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department VARCHAR(50),
    salary DECIMAL(10, 2),
    hire_date DATE
);

INSERT INTO employees VALUES
(1, 'John', 'Doe', 'IT', 75000, '2020-01-15'),
(2, 'Jane', 'Smith', 'HR', 65000, '2019-05-11'),
(3, 'Bob', 'Johnson', 'IT', 80000, '2018-03-23'),
(4, 'Alice', 'Williams', 'Finance', 72000, '2021-09-30'),
(5, 'Charlie', 'Brown', 'IT', 68000, '2022-02-14'),
(6, 'Eva', 'Davis', 'HR', 61000, '2020-11-18'),
(7, 'Frank', 'Miller', 'Finance', 79000, '2017-07-12'),
(8, 'Grace', 'Taylor', 'IT', 77000, '2019-04-22'),
(9, 'Henry', 'Anderson', 'Finance', 71000, '2021-01-05'),
(10, 'Ivy', 'Thomas', 'HR', 63000, '2022-06-30');

CREATE TABLE projects (
    project_id INT PRIMARY KEY,
    project_name VARCHAR(100),
    start_date DATE,
    end_date DATE
);

INSERT INTO projects VALUES
(1, 'Database Migration', '2023-01-01', '2023-06-30'),
(2, 'New HR System', '2023-03-15', '2023-12-31'),
(3, 'Financial Reporting Tool', '2023-02-01', '2023-11-30'),
(4, 'IT Infrastructure Upgrade', '2023-05-01', '2024-04-30');

CREATE TABLE employee_projects (
    employee_id INT,
    project_id INT,
    role VARCHAR(50),
    FOREIGN KEY (employee_id) REFERENCES employees(employee_id),
    FOREIGN KEY (project_id) REFERENCES projects(project_id)
);

INSERT INTO employee_projects VALUES
(1, 1, 'Project Lead'),
(2, 2, 'Project Manager'),
(3, 1, 'Database Admin'),
(4, 3, 'Financial Analyst'),
(5, 4, 'Network Engineer'),
(6, 2, 'HR Specialist'),
(7, 3, 'Data Analyst'),
(8, 4, 'Systems Architect'),
(1, 4, 'Security Consultant'),
(3, 4, 'Software Developer');
```

```sql
-- Questions

-- 1. Write a query to find the top 3 highest paid employees in each department.
select department,first_name,last_name,salary, rankno
from (
select department, first_name,last_name,salary, dense_rank() over (partition by department order by salary desc) as rankno
from employees
)ranked_employees
where rankno <=3
order by department , rankno;


-- 2. Calculate the running total of salaries in each department, ordered by hire date.
SELECT department, first_name, hire_date, salary,
       SUM(salary) OVER (PARTITION BY department ORDER BY hire_date) AS running_total
FROM employees
ORDER BY department, hire_date;

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

-- 4. Identify projects that have team members from all departments.
SELECT p.project_name
FROM employee_projects ep
JOIN employees e ON ep.employee_id = e.employee_id
JOIN projects p ON ep.project_id = p.project_id
GROUP BY p.project_id, p.project_name
HAVING COUNT(DISTINCT e.department) = (SELECT COUNT(DISTINCT department) FROM employees);

-- 5. Calculate the average salary for each department, but only include employees hired in the last 3 years.
SELECT department, AVG(salary) AS avg_salary
FROM employees
WHERE hire_date >= DATE_SUB(CURDATE(), INTERVAL 3 YEAR)
GROUP BY department;

-- 6. Create a pivot table showing the count of employees in each department, with columns for different salary ranges.
SELECT department,
       SUM(CASE WHEN salary < 65000 THEN 1 ELSE 0 END) AS '<65000',
       SUM(CASE WHEN salary BETWEEN 65000 AND 75000 THEN 1 ELSE 0 END) AS '65000-75000',
       SUM(CASE WHEN salary > 75000 THEN 1 ELSE 0 END) AS '>75000'
FROM employees
GROUP BY department;

-- 7. Find the employee(s) with the highest salary in their respective departments, who are also working on the longest-running project.
WITH project_duration AS (
    SELECT project_id, DATEDIFF(end_date, start_date) AS duration
    FROM projects
),
max_salary_employee AS (
    SELECT department, MAX(salary) AS max_salary
    FROM employees
    GROUP BY department
)
SELECT e.first_name, e.department, e.salary, p.project_name
FROM employees e
JOIN employee_projects ep ON e.employee_id = ep.employee_id
JOIN project_duration pd ON ep.project_id = pd.project_id
JOIN projects p ON p.project_id = pd.project_id
JOIN max_salary_employee mse ON e.department = mse.department AND e.salary = mse.max_salary
ORDER BY pd.duration DESC
LIMIT 1;
```

```sql
-- 8. Calculate the percentage of each department's salary compared to the total salary of the company.
SELECT department,
       SUM(salary) AS total_department_salary,
       (SUM(salary) / (SELECT SUM(salary) FROM employees) * 100) AS percentage_of_total
FROM employees
GROUP BY department
order by percentage_of_total desc;

-- 9. Identify employees who have a higher salary than their department's average, and show by what percentage their salary exceeds the average.
WITH avg_salary_per_department AS (
    SELECT department, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department
)
SELECT e.first_name, e.department, e.salary,
       ((e.salary - a.avg_salary) / a.avg_salary) * 100 AS percentage_above_avg
FROM employees e
JOIN avg_salary_per_department a ON e.department = a.department
WHERE e.salary > a.avg_salary;

-- 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.
SELECT e.first_name, p.project_name, ep.role
FROM employees e
JOIN employee_projects ep ON e.employee_id = ep.employee_id
JOIN projects p ON ep.project_id = p.project_id
ORDER BY e.first_name, p.project_name;



-- 11. Implement a query to find the "Kevin Bacon Number" equivalent for projects.
WITH RECURSIVE project_connections AS (
    SELECT ep1.employee_id as employee1, ep2.employee_id as employee2, 1 as connection_level
    FROM employee_projects ep1
    JOIN employee_projects ep2 ON ep1.project_id = ep2.project_id AND ep1.employee_id < ep2.employee_id
    UNION
    SELECT pc.employee1, ep.employee_id as employee2, pc.connection_level + 1
    FROM project_connections pc
    JOIN employee_projects ep ON pc.employee2 = ep.employee_id
    WHERE pc.employee1 < ep.employee_id AND pc.connection_level < 6
)
SELECT e1.first_name || ' ' || e1.last_name as employee1, e2.first_name || ' ' || e2.last_name as employee2, MIN(connection_level) as shortest_connection
FROM project_connections pc
JOIN employees e1 ON pc.employee1 = e1.employee_id
JOIN employees e2 ON pc.employee2 = e2.employee_id
GROUP BY e1.employee_id, e1.first_name, e1.last_name, e2.employee_id, e2.first_name, e2.last_name
ORDER BY e1.last_name, e1.first_name, shortest_connection;
```

EXPLAIN output:

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 | filtered | Extra       |
+----+-------------+-------+------------+--------+---------------+---------+---------+---------
------------+------+---------+-------------+
|  1 | SIMPLE      | e     | NULL       | ALL    | NULL          | NULL    | NULL    |
NULL                 | 1000 |    33.33 | Using where |
|  1 | SIMPLE      | d     | NULL       | eq_ref | 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.

# DAY 17 (9 October)

## Database Management System (DBMS)

**A database is a collection of interrelated data that helps in the efficient retrieval, insertion, and deletion of data from the database and organizes the data in the form of tables, views, schemas, reports, etc. For Example, a university database organizes the data about students, faculty, admin staff, etc. which helps in the efficient retrieval, insertion, and deletion of data from it.**

**A Database Management System (DBMS) is a software system that is designed to manage and organize data in a structured manner. It allows users to create, modify, and query a database, as well as manage the security and access controls for that database. DBMS provides an environment to store and retrieve data in convenient and efficient manner.**

## Types of Keys in Relational Model (Candidate, Super, Primary, Alternate and Foreign)

### Different Types of Database Keys

**Candidate Key**:

A set of attributes that uniquely identify a record in a table. Multiple candidate keys can exist, and one is chosen as the primary key.

**Primary Key**:

A specific candidate key chosen to uniquely identify all records in a table. It must have unique, non-null values.

**Super Key**:

A set of attributes that can uniquely identify a record. A super key may contain additional attributes that are not necessary for unique identification.

**Alternate Key**:

Candidate keys that are not chosen as the primary key. They can still uniquely identify records but are not the main key.

**Foreign Key**:

A key in one table that refers to the primary key in another table, establishing a relationship between the two tables.

**Composite Key**:

A key that consists of two or more attributes that together uniquely identify a record, where no individual attribute alone can provide uniqueness.

# Introduction of Database Normalization

Normalization is an important process in database design that helps improve the database's efficiency, consistency, and accuracy. It makes it easier to manage and maintain the data and ensures that the database is adaptable to changing business needs.

## What is Database Normalization?

Database normalization is the process of organizing the attributes of the database to reduce or eliminate data redundancy (having the same data but at different places). Data redundancy unnecessarily increases the size of the database as the same data is repeated in many places. Inconsistency problems also arise during insert, delete, and update operations.

## Normalization of DBMS

In database management systems (DBMS), normal forms are a series of guidelines that help to ensure that the design of a database is efficient, organized, and free from data anomalies. There are several levels of normalization, each with its own set of guidelines, known as normal forms.

### Important Points Regarding Normal Forms in DBMS

- **First Normal Form (1NF):** This is the most basic level of normalization. In 1NF, each table cell should contain only a single value, and each column should have a unique name. The first normal form helps to eliminate duplicate data and simplify queries.
- **Second Normal Form (2NF):** 2NF eliminates redundant data by requiring that each non-key attribute be dependent on the primary key. This means that each column should be directly related to the primary key, and not to other columns.
- **Third Normal Form (3NF):** 3NF builds on 2NF by requiring that all non-key attributes are independent of each other. This means that each column

should be directly related to the primary key, and not to any other columns in the same table.

- **Boyce-Codd Normal Form (BCNF):** BCNF is a stricter form of 3NF that ensures that each determinant in a table is a candidate key. In other words, BCNF ensures that each non-key attribute is dependent only on the candidate key.
- **Fourth Normal Form (4NF):** 4NF is a further refinement of BCNF that ensures that a table does not contain any multi-valued dependencies.
- **Fifth Normal Form (5NF):** 5NF is the highest level of normalization and involves decomposing a table into smaller tables to remove data redundancy and improve data integrity.

Normal forms help to reduce data redundancy, increase data consistency, and improve database performance. However, higher levels of normalization can lead to more complex database designs and queries. It is important to strike a balance between normalization and practicality when designing a database.

## SQL Practice:

```sql
CREATE database DB9;
USE DB9;
show tables;

-- Create Tables
CREATE TABLE customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    registration_date DATE
);

CREATE TABLE products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    product_name VARCHAR(100),
    category VARCHAR(50),
    price DECIMAL(10, 2),
    stock_quantity INT
);

CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    order_date DATETIME,
    total_amount DECIMAL(10, 2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

CREATE TABLE order_items (
    order_item_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    product_id INT,
    quantity INT,
    price_per_unit DECIMAL(10, 2),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

CREATE TABLE product_reviews (
    review_id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT,
    customer_id INT,
    rating INT CHECK (rating BETWEEN 1 AND 5),
    review_text TEXT,
    review_date DATE,
    FOREIGN KEY (product_id) REFERENCES products(product_id),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

-- Insert into customers
INSERT INTO customers (first_name, last_name, email, registration_date)
VALUES
('John', 'Doe', 'john.doe@example.com', '2023-01-15'),
('Jane', 'Smith', 'jane.smith@example.com', '2023-02-20'),
('Alice', 'Johnson', 'alice.johnson@example.com', '2023-03-05'),
('Bob', 'Williams', 'bob.williams@example.com', '2023-04-10'),
('Charlie', 'Brown', 'charlie.brown@example.com', '2023-05-15'),
('Diana', 'Evans', 'diana.evans@example.com', '2023-06-20'),
('Emily', 'Clark', 'emily.clark@example.com', '2023-07-25'),
('Frank', 'Miller', 'frank.miller@example.com', '2023-08-30');

-- Insert into products
INSERT INTO products (product_name, category, price, stock_quantity)
VALUES
('Laptop', 'Electronics', 999.99, 50),
('Smartphone', 'Electronics', 599.99, 100),
('Headphones', 'Accessories', 199.99, 150),
('Tablet', 'Electronics', 299.99, 75),
('Smartwatch', 'Electronics', 149.99, 120),
('Keyboard', 'Accessories', 49.99, 200),
('Monitor', 'Electronics', 199.99, 80),
('Mouse', 'Accessories', 29.99, 300);

-- Insert into orders
INSERT INTO orders (customer_id, order_date, total_amount)
VALUES
(1, '2024-10-01 10:00:00', 1599.98),
(2, '2024-10-03 12:30:00', 599.99),
(3, '2024-10-04 15:45:00', 299.99),
(4, '2024-10-05 17:20:00', 179.98),
(5, '2024-10-06 09:10:00', 149.99),
(6, '2024-10-07 13:55:00', 199.99),
(1, '2024-10-08 11:30:00', 49.99);

-- Insert into order_items
INSERT INTO order_items (order_id, product_id, quantity, price_per_unit)
VALUES
(1, 1, 1, 999.99),
(1, 2, 1, 599.99),
(2, 2, 1, 599.99),
(3, 4, 1, 299.99),
(4, 3, 2, 49.99),
(5, 5, 1, 149.99),
(6, 6, 1, 199.99),
(7, 7, 1, 49.99);

-- Insert into product_reviews
INSERT INTO product_reviews (product_id, customer_id, rating, review_text, review_date)
VALUES
(1, 1, 5, 'Great laptop, very satisfied with the performance!', '2024-10-05'),
(2, 2, 4, 'Good smartphone, but battery life could be better.', '2024-10-06'),
(3, 1, 3, 'Average headphones, decent sound quality but uncomfortable.', '2024-10-07'),
(4, 3, 5, 'Amazing monitor for gaming!', '2024-10-08'),
(5, 4, 4, 'Decent tablet for the price.', '2024-10-09'),
(6, 5, 3, 'Smartwatch is okay, but slow sometimes.', '2024-10-09'),
(7, 6, 5, 'Fantastic keyboard for typing!', '2024-10-09'),
(8, 1, 4, 'Good mouse, comfortable for long use.', '2024-10-09');
```

```sql
-- Advanced SQL Practice Questions

-- Question 1: Find the top 3 customers who have spent the most money,
-- along with their total spend and the number of orders they've made.

select * from customers
join orders on customers.customer_id = orders.customer_id
order by total_amount desc
limit 3;

-- Question 2: Calculate the average rating for each product category,
-- but only include categories with at least 2 reviews.

select p.category, avg(pr.rating) as average_rating
from products p
join product_reviews pr
on p.product_id = pr.product_id
group by p.category
having count(pr.review_id)>=2;


-- Question 3: Find products that have never been ordered.

select p.product_name, p.product_id
from products p
left join order_items oi
on p.product_id = oi.product_id
where oi.product_id is null;

-- 4. For each customer, find the time difference between their
-- registration date and their first order date.

SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    c.registration_date,
    MIN(o.order_date) AS first_order_date,
    DATEDIFF(MIN(o.order_date), c.registration_date) AS days_to_first_order
FROM
    customers c
LEFT JOIN
    orders o ON c.customer_id = o.customer_id
GROUP BY
    c.customer_id, c.first_name, c.last_name, c.registration_date
ORDER BY
    days_to_first_order;

-- 5. Create a report showing the total revenue for each month,
-- along with a running total of revenue throughout the year.

SELECT
    DATE_FORMAT(o.order_date, '%Y-%m') AS month,
    SUM(oi.quantity * oi.price_per_unit) AS monthly_revenue,
    SUM(SUM(oi.quantity * oi.price_per_unit)) OVER (
        ORDER BY DATE_FORMAT(o.order_date, '%Y-%m')
    ) AS running_total
FROM
    orders o
JOIN
    order_items oi ON o.order_id = oi.order_id
GROUP BY
    DATE_FORMAT(o.order_date, '%Y-%m')
ORDER BY
    month;
```

```sql
-- 6. Identify customers who have made a purchase but have never left a product review.
SELECT DISTINCT
    c.customer_id,
    c.first_name,
    c.last_name
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
LEFT JOIN product_reviews pr ON c.customer_id = pr.customer_id
WHERE pr.review_id IS NULL;

-- 7. Find the product that has been ordered the most times (by quantity).
SELECT
    p.product_id,
    p.product_name,
    SUM(oi.quantity) AS total_quantity_ordered
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_id, p.product_name
ORDER BY total_quantity_ordered DESC
LIMIT 1;

-- 8. Calculate the percentage of total revenue that each product category contributes.
WITH category_revenue AS (
    SELECT
        c.category_id,
        c.category_name,
        SUM(oi.quantity * oi.price_per_unit) AS revenue
    FROM categories c
    JOIN products p ON c.category_id = p.category_id
    JOIN order_items oi ON p.product_id = oi.product_id
    GROUP BY c.category_id, c.category_name
),
total_revenue AS (
    SELECT SUM(revenue) AS total FROM category_revenue
)
SELECT
    cr.category_name,
    cr.revenue,
    (cr.revenue / tr.total) * 100 AS percentage_of_total
FROM category_revenue cr
CROSS JOIN total_revenue tr
ORDER BY percentage_of_total DESC;

-- 9. For each customer, find their most frequently purchased product category.
WITH customer_category_purchases AS (
    SELECT
        o.customer_id,
        p.category_id,
        COUNT(*) AS purchase_count,
        ROW_NUMBER() OVER (
            PARTITION BY o.customer_id
            ORDER BY COUNT(*) DESC
        ) AS rn
    FROM orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    JOIN products p ON oi.product_id = p.product_id
    GROUP BY o.customer_id, p.category_id
)
SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    cat.category_name AS most_frequent_category,
    ccp.purchase_count
FROM customer_category_purchases ccp
JOIN customers c ON ccp.customer_id = c.customer_id
JOIN categories cat ON ccp.category_id = cat.category_id
WHERE ccp.rn = 1;

-- 10. Create a query to show the distribution of ratings (count of 1-star, 2-star, etc.) for each product.
SELECT
    p.product_id,
    p.product_name,
    SUM(CASE WHEN pr.rating = 1 THEN 1 ELSE 0 END) AS one_star,
    SUM(CASE WHEN pr.rating = 2 THEN 1 ELSE 0 END) AS two_star,
    SUM(CASE WHEN pr.rating = 3 THEN 1 ELSE 0 END) AS three_star,
    SUM(CASE WHEN pr.rating = 4 THEN 1 ELSE 0 END) AS four_star,
    SUM(CASE WHEN pr.rating = 5 THEN 1 ELSE 0 END) AS five_star
FROM products p
LEFT JOIN product_reviews pr ON p.product_id = pr.product_id
GROUP BY p.product_id, p.product_name
ORDER BY p.product_id;
```

## SQL PROBLEM 1:

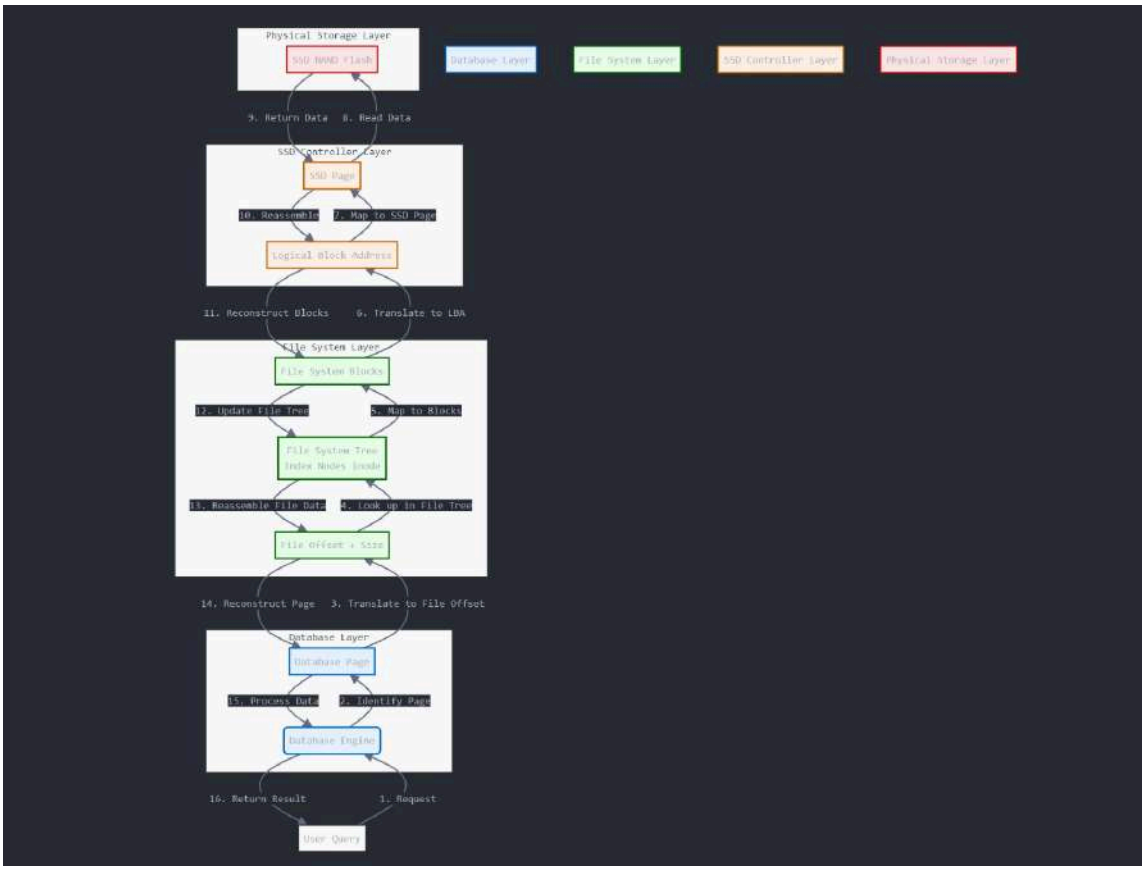
```sql
-- Create a sample table for stock prices
CREATE TABLE stock_prices (
    date DATE,
    stock_symbol VARCHAR(10),
    closing_price DECIMAL(10, 2)
);

-- Insert sample data
INSERT INTO stock_prices (date, stock_symbol, closing_price) VALUES
('2023-01-01', 'AAPL', 150.00),
('2023-01-02', 'AAPL', 152.50),
('2023-01-03', 'AAPL', 151.75),
('2023-01-04', 'AAPL', 153.00),
('2023-01-05', 'AAPL', 155.25),
('2023-01-01', 'GOOGL', 2800.00),
('2023-01-02', 'GOOGL', 2825.00),
('2023-01-03', 'GOOGL', 2815.50),
('2023-01-04', 'GOOGL', 2830.00),
('2023-01-05', 'GOOGL', 2850.75);


select date,
stock_symbol,
closing_price,
lead(closing_price,2) over (partition by stock_symbol order by date) as next_day_price
from stock_prices
order by stock_symbol, date;



select date,
stock_symbol,
lag(closing_price,1) over (partition by stock_symbol order by date) as previous_day_price
,closing_price
from stock_prices
order by stock_symbol, date;



select date,
stock_symbol,
closing_price,
lag(closing_price,1) over (partition by stock_symbol order by date) as previous_day_price
,closing_price-LAG(closing_price,1) over (partition by stock_symbol order by date) as price_change
from stock_prices
order by stock_symbol, date;
```

## SQL PROBLEM 2:

```sql
CREATE database D010;
USE D010;

-- Create tables
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    unit_price DECIMAL(10, 2)
);

CREATE TABLE sales (
    sale_id INT PRIMARY KEY,
    product_id INT,
    sale_date DATE,
    quantity INT,
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

-- Insert sample data
INSERT INTO products (product_id, product_name, category, unit_price) VALUES
(1, 'Laptop', 'Electronics', 999.99),
(2, 'Smartphone', 'Electronics', 599.99),
(3, 'Tablet', 'Electronics', 399.99),
(4, 'Desk Chair', 'Furniture', 149.99),
(5, 'Coffee Table', 'Furniture', 199.99),
(6, 'Bookshelf', 'Furniture', 89.99),
(7, 'Running Shoes', 'Apparel', 79.99),
(8, 'T-shirt', 'Apparel', 19.99),
(9, 'Jeans', 'Apparel', 59.99);

INSERT INTO sales (sale_id, product_id, sale_date, quantity) VALUES
(1, 1, '2023-01-15', 2),
(2, 2, '2023-01-16', 3),
(3, 4, '2023-01-17', 1),
(4, 7, '2023-01-18', 4),
(5, 3, '2023-02-01', 2),
(6, 5, '2023-02-02', 1),
(7, 8, '2023-02-03', 5),
(8, 1, '2023-02-15', 1),
(9, 6, '2023-02-16', 2),
(10, 2, '2023-02-17', 2),
(11, 9, '2023-03-01', 3),
(12, 4, '2023-03-02', 2),
(13, 7, '2023-03-03', 3),
(14, 3, '2023-03-15', 1),
(15, 5, '2023-03-16', 1);

-- Problem Statement:
-- Analyze the sales data to find the top-performing product category for each month of 2023.
-- The performance should be based on total revenue (quantity sold * unit price).
-- Your query should return the month, the top-performing category, and its total revenue.
-- In case of a tie, include all categories with the same top performance.

-- Expected output format:
-- | month | top_category | total_revenue |
-- |-------|--------------|---------------|
-- | 2023-01 | Electronics | 2799.97      |
-- | 2023-03 | Furniture   | 509.98       |
-- | 2023-03 | Electronics | 999.98       |

-- Note: The actual values may differ based on the sample data provided.

SELECT year, month, category, total_revenue
FROM (
    SELECT
        YEAR(s.sale_date) AS year,
        MONTH(s.sale_date) AS month,
        p.category AS category,
        SUM(s.quantity * p.unit_price) AS total_revenue
    FROM sales s
    JOIN products p ON s.product_id = p.product_id
    GROUP BY year, month, p.category
) AS monthly_revenue
WHERE total_revenue = (
    SELECT MAX(total_revenue)
    FROM (
        SELECT
            YEAR(s2.sale_date) AS year,
            MONTH(s2.sale_date) AS month,
            p2.category AS category,
            SUM(s2.quantity * p2.unit_price) AS total_revenue
        FROM sales s2
        JOIN products p2 ON s2.product_id = p2.product_id
        GROUP BY year, month, p2.category
    ) AS category_revenue
    WHERE category_revenue.year = monthly_revenue.year
    AND category_revenue.month = monthly_revenue.month
)
ORDER BY year, month;
```

# DAY 18 (10 October)

```sql
CREATE database DB11;
USE DB11;

create table if not exists large_table(
id int auto_increment primary key,
name varchar(50),
value INT
);

DELIMITER //
CREATE procedure INSERT_MILLION_RECORDS()
BEGIN
  declare i int default 0;
  while i <100 do
      insert into large_table(name,value) values(concat('Name',i),floor(1 + RAND()*100));
      set i=i+1;
  end while;
END //

DELIMITER ;
-- Drop procedure INSERT_MILLION_RECORDS;

SHOW procedure status WHERE DB ='DB11';
-- Drop table large_table;

select count(*) from large_table;
CALL INSERT_MILLION_RECORDS();

select sql_no_cache sum(value) from large_table;
select  sum(value) from large_table;

create index idx_value on large_table(value);
alter table large_table drop index idx_value;
select * from large_table;

select a.value
from large_table a
cross join large_table b
on a.value=b.value*100;
```




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.

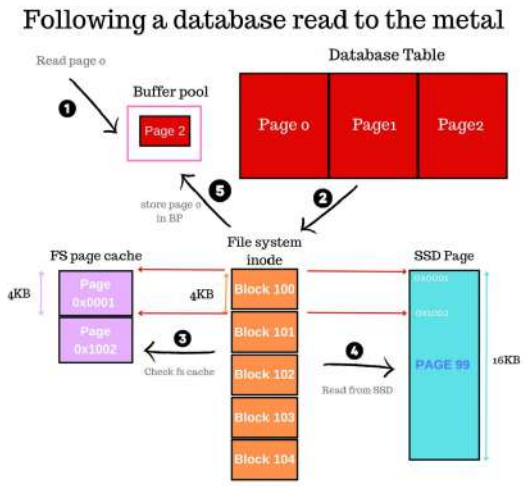
Why is this system used?

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.

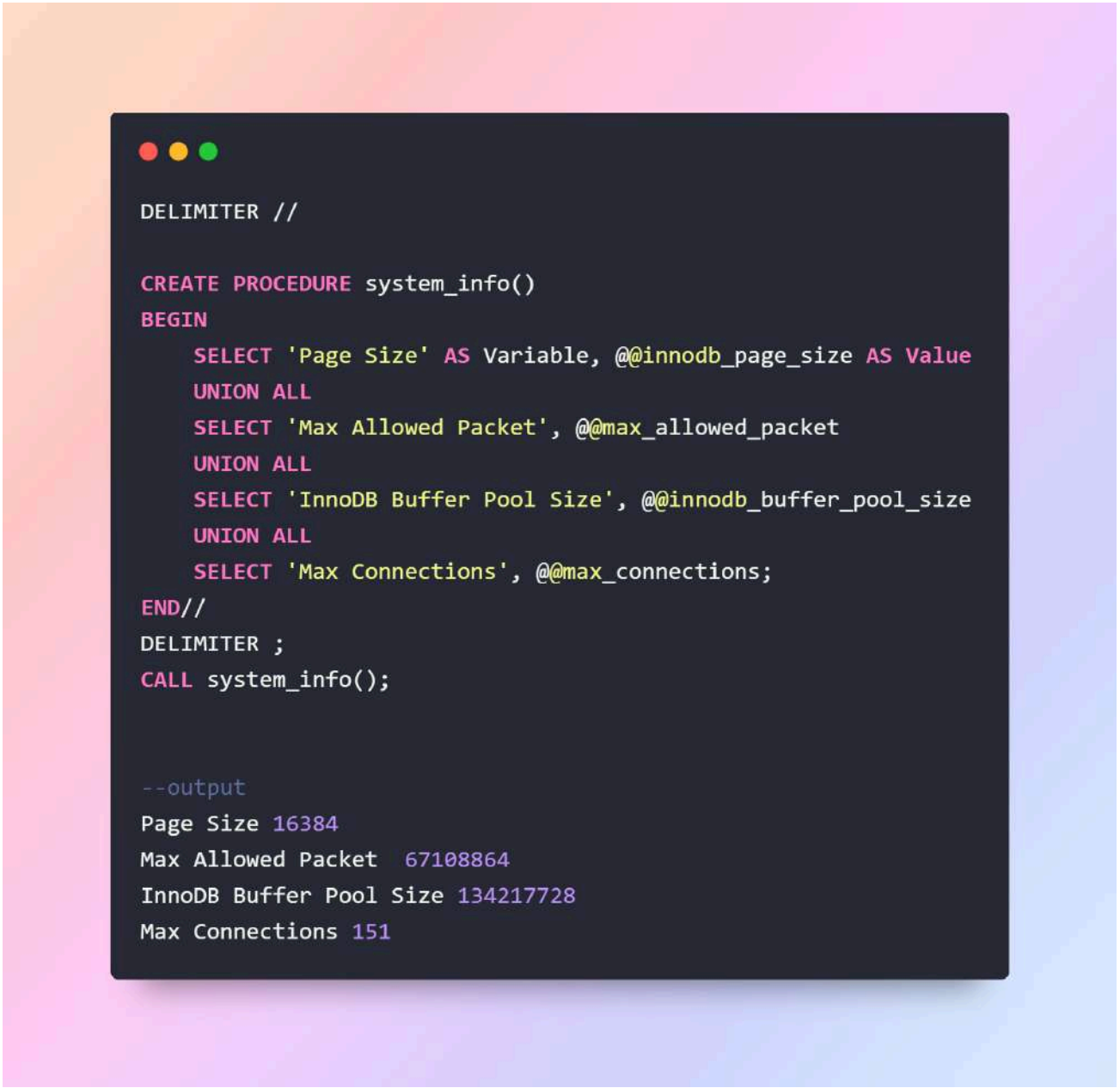
In computers:

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.



## System checks :



```
DELIMITER //

CREATE PROCEDURE system_info()
BEGIN
    SELECT 'Page Size' AS Variable, @@innodb_page_size AS Value
    UNION ALL
    SELECT 'Max Allowed Packet', @@max_allowed_packet
    UNION ALL
    SELECT 'InnoDB Buffer Pool Size', @@innodb_buffer_pool_size
    UNION ALL
    SELECT 'Max Connections', @@max_connections;
END//
DELIMITER ;
CALL system_info();


--output
Page Size 16384
Max Allowed Packet  67108864
InnoDB Buffer Pool Size 134217728
Max Connections 151
```

```sql
CREATE database DB12;
USE DB12;

create table customers(
customer_id int primary key,
name varchar(100),
email varchar(100),
registration date);

--  Populate Customers table
INSERT INTO Customers (Customer_ID, Name, Email, registration)
VALUES
(1, 'John Doe', 'john.doe@email.com', '2024-09-15'),
(2, 'Jane Smith', 'jane.smith@email.com', '2024-09-20'),
(3, 'Bob Johnson', 'bob.johnson@email.com', '2024-09-25'),
(4, 'Alice Brown', 'alice.brown@email.com', '2024-09-30'),
(5, 'Charlie Davis', 'charlie.davis@email.com', '2024-10-01'),
(6, 'Eva Wilson', 'eva.wilson@email.com', '2024-10-02'),
(7, 'Frank Miller', 'frank.miller@email.com', '2024-10-03'),
(8, 'Grace Lee', 'grace.lee@email.com', '2024-10-04'),
(9, 'Henry Taylor', 'henry.taylor@email.com', '2024-10-05'),
(10, 'Ivy Clark', 'ivy.clark@email.com', '2024-10-06');

select * from customers;
create index idx_customer_reg_Date on customers(registration);
update customers set registration='2024-0-20';

set session join_buffer_size=4194304;
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);

-- Insert 90 more customers with registration dates spread over the last 60 days
INSERT INTO Customers (Customer_ID, Name, Email, registration)
SELECT
  10 + num,
  CONCAT('Customer', num),
  CONCAT('customer', num, '@email.com'),
  DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND() * 60) DAY)
FROM (
  SELECT a.N + b.N * 10 + 1 as num
  FROM
    (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 SELECT 9) a,
    (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
  ORDER BY num
  LIMIT 90
) numbers;
create table orders(orderid int primary key,
customer_id int,
orderdate date,
totalamount decimal(10,2),
foreign key(customer_id) references customers(customer_id));

-- Populate Orders table
INSERT INTO Orders (OrderID, customer_id, OrderDate, TotalAmount)
VALUES
(1, 1, '2024-09-16', 150.00),
(2, 2, '2024-09-21', 200.50),
(3, 3, '2024-09-26', 75.25),
(4, 4, '2024-10-01', 300.75),
(5, 5, '2024-10-02', 50.00),
(6, 6, '2024-10-03', 125.50),
(7, 7, '2024-10-04', 80.00),
(8, 8, '2024-10-05', 220.25),
(9, 9, '2024-10-06', 175.00),
(10, 10, '2024-10-07', 90.50);

-- Insert 190 more orders with random customers and dates
INSERT INTO Orders (OrderID, Customer_ID, OrderDate, TotalAmount)
SELECT
  10 + num,
  1 + FLOOR(RAND() * 100),
  DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND() * 30) DAY),
  ROUND(50 + RAND() * 450, 2)
FROM (
  SELECT a.N + b.N * 10 + c.N * 100 + 1 AS num
  FROM
    (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 SELECT 9) a,
    (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 SELECT 9) b,
    (SELECT 0 AS N UNION SELECT 1) c
  ORDER BY num
  LIMIT 190
) numbers;
```

# DAY 19 (11 October)

## 1. Database Design

- The process of defining the structure of a database to ensure the organization of data is efficient and supports the needs of the system.
- Focuses on tables, relationships, normalization, and indexes.
- It includes creating entity-relationship diagrams (ERDs) to model how data entities relate to each other.

## 2. Database Modeling

- Involves designing the structure of a database that defines entities, attributes, and relationships.
- Models help visualize database structure.

### a) Star Schema

- Simplest data warehouse schema.
- Contains a central fact table connected to dimension tables (denormalized).
- Fact table stores measures (e.g., sales data).
- Dimension tables provide context (e.g., date, location).

### b) Snowflake Schema

- Similar to star schema but with normalized dimension tables.
- Dimension tables are split into additional related tables, reducing redundancy.
- More complex than star schema but saves storage space.

### c) Galaxy Schema (Fact Constellation)

- A complex schema with multiple fact tables sharing dimension tables.
- Used in cases with multiple business processes or subject areas.
- Combines multiple star or snowflake schemas.

## 3. Data Warehousing

- A system used to store, retrieve, and analyze large volumes of data from different sources.
- Optimized for query processing, reporting, and analysis.
- Follows ETL (Extract, Transform, Load) processes to collect data from multiple sources and load it into the warehouse.

## 4. Slowly Changing Dimension (SCD)

- Techniques to manage changes in dimension data over time.
- Types:
    - Type 1: Overwrites old data with new data (no history).
    - Type 2: Stores multiple records for each changing attribute (keeps history).
    - Type 3: Adds a new column to store the previous data value (limited history).

## 5. Junk Dimension

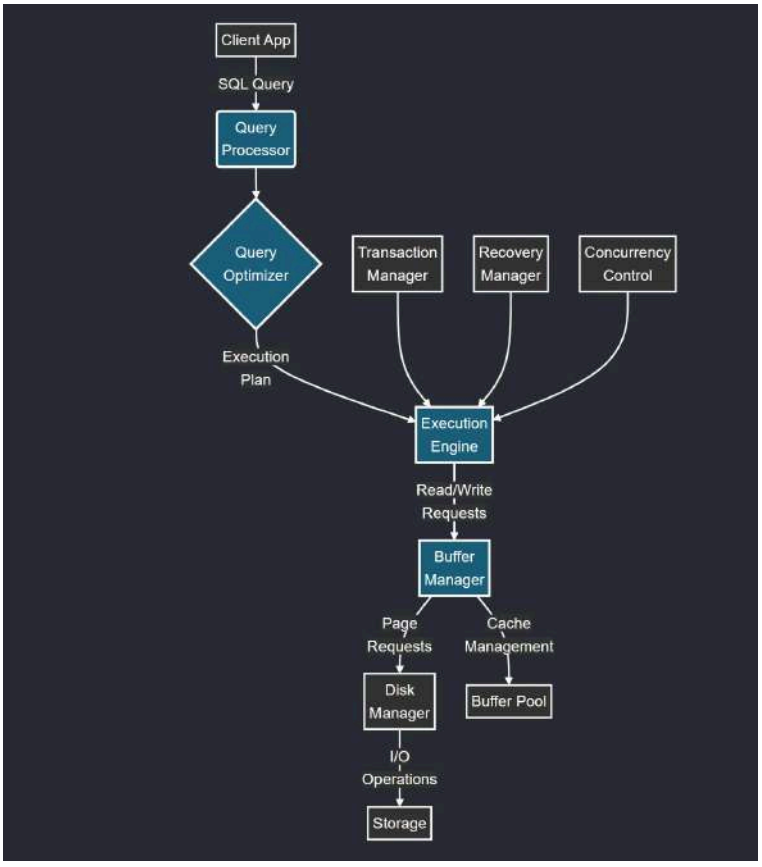
- A dimension created by combining several low-cardinality attributes that do not belong to any specific dimension.
- Helps reduce clutter by grouping unnecessary flags or indicators into one dimension.

## 6. Degenerate Dimension

- A dimension in a fact table that does not have its own dimension table.
- Typically contains information like order numbers, transaction IDs, etc., which don’t need separate tables but are required for analysis.

## Database Anatomy: The Inner Workings

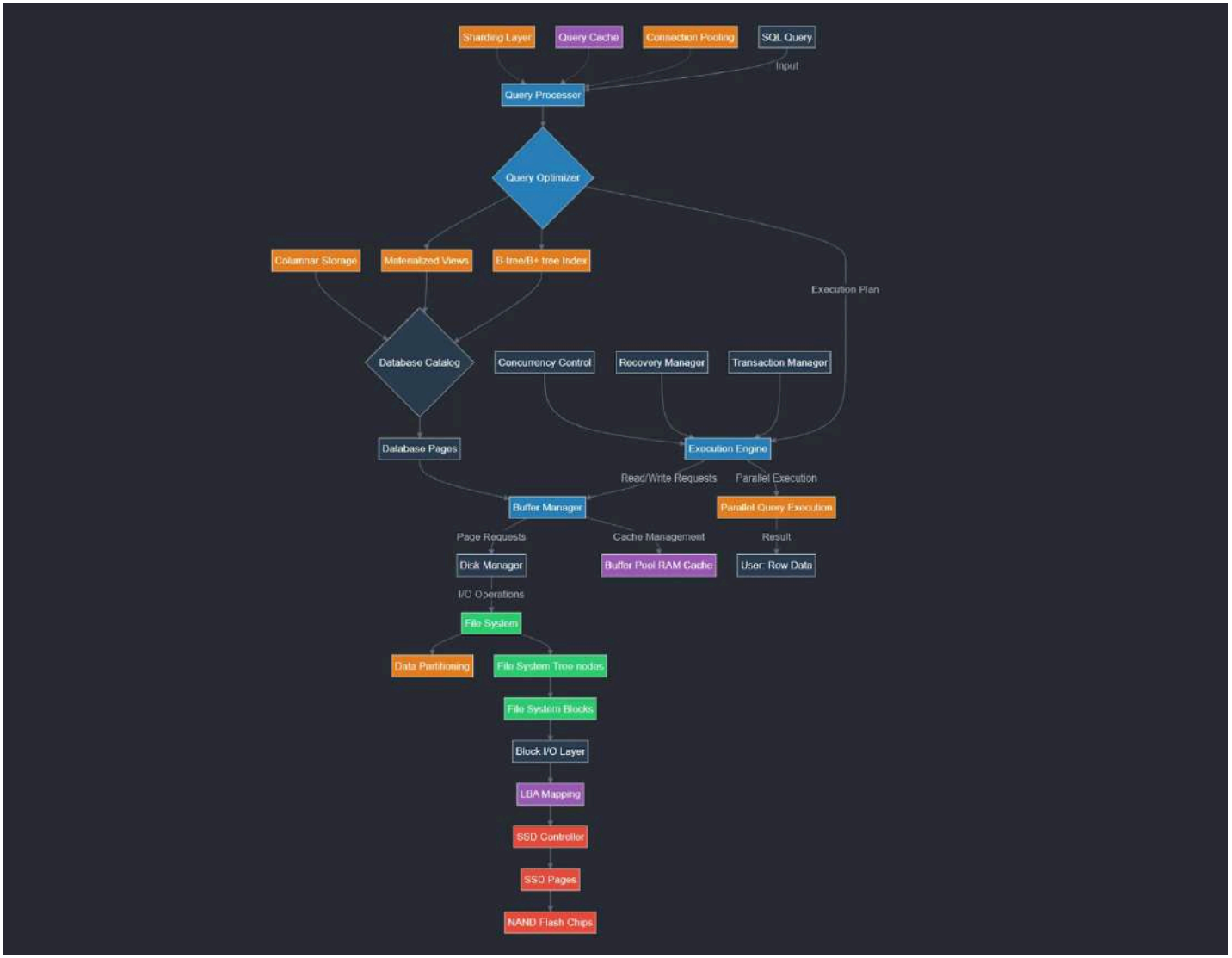
### Now, let's take a closer look at the internal structure of a database system:




## The Journey of Your Data Request

User Query Our journey begins with a user sending a SQL query:

```
SELECT * FROM users WHERE id = 1008
```


Sharding Layer
Query Cache
Connection Pooling
SQL Query
Input
Query Processor
Query Optimizer
Columnar Storage
Materialized Views
B-tree/B+ tree Index
Execution Plan
Database Catalog
Concurrency Control
Recovery Manager
Transaction Manager
Database Pages
Execution Engine
Read/Write Requests
Parallel Execution
Buffer Manager
Parallel Query Execution
Page Requests
Cache Management
Result
Disk Manager
Buffer Pool RAM Cache
User: Row Data
I/O Operations
File System
Data Partitioning
File System Tree nodes
File System Blocks
Block I/O Layer
LBA Mapping
SSD Controller
SSD Pages
NAND Flash Chips

```sql
-- ## Use Case 1: E-commerce Order Analytics

-- ### Problem Statement:
-- An e-commerce platform with 10 million orders needs to analyze order trends, product performance, and customer behavior.

-- ### Normalized Schema:
-- sql
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100),
    Address VARCHAR(200)
);

CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    Name VARCHAR(100),
    Category VARCHAR(50),
    Price DECIMAL(10, 2)
);

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

CREATE TABLE OrderItems (
    OrderItemID INT PRIMARY KEY,
    OrderID INT,
    ProductID INT,
    Quantity INT,
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

-- ### Slow Query:
-- sql
SELECT p.Category, YEAR(o.OrderDate) AS Year, SUM(p.Price * oi.Quantity) AS TotalSales
FROM Orders o
JOIN OrderItems oi ON o.OrderID = oi.OrderID
JOIN Products p ON oi.ProductID = p.ProductID
GROUP BY p.Category, YEAR(o.OrderDate)
ORDER BY Year, TotalSales DESC;

-- ### Star Schema Solution:
-- sql
CREATE TABLE DimDate (
    DateID INT PRIMARY KEY,
    Date DATE,
    Year INT,
    Month INT
);

CREATE TABLE DimProduct (
    ProductID INT PRIMARY KEY,
    Name VARCHAR(100),
    Category VARCHAR(50)
);

CREATE TABLE DimCustomer (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100)
);

CREATE TABLE FactSales (
    SalesID INT PRIMARY KEY,
    DateID INT,
    ProductID INT,
    CustomerID INT,
    Quantity INT,
    TotalAmount DECIMAL(10, 2),
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),
    FOREIGN KEY (ProductID) REFERENCES DimProduct(ProductID),
    FOREIGN KEY (CustomerID) REFERENCES DimCustomer(CustomerID)
);

-- ### Optimized Query:
-- sql
SELECT dp.Category, dd.Year, SUM(fs.TotalAmount) AS TotalSales
FROM FactSales fs
JOIN DimProduct dp ON fs.ProductID = dp.ProductID
JOIN DimDate dd ON fs.DateID = dd.DateID
GROUP BY dp.Category, dd.Year
ORDER BY dd.Year, TotalSales DESC;
```

## SQL Triggers

### Definition:

- A trigger is a special type of stored procedure in SQL that automatically executes or fires in response to specific events on a table, such as `INSERT`, `UPDATE`, or `DELETE`.

### Key Features:

- Automatic Execution: Triggers run automatically when the specified event occurs, without needing explicit calls from application code.
- Event-Driven: Triggers are associated with specific events (e.g., before or after a data modification operation).
- Conditional Logic: They can include conditional logic to control whether the trigger action executes based on the data being modified.

### Types of Triggers:

1. BEFORE Trigger:
    - Executes before the triggering event occurs.
    - Often used to validate or modify data before it is committed to the database.
2. AFTER Trigger:
    - Executes after the triggering event occurs.
    - Commonly used for actions like logging changes, enforcing business rules, or updating related tables.
3. INSTEAD OF Trigger:
    - Executes instead of the triggering event.
    - Mainly used for views, allowing operations on a view to affect the underlying tables.

### Common Use Cases:

- Auditing Changes: Tracking modifications to a table by recording changes in an audit table.
- Enforcing Business Rules: Validating data conditions before allowing modifications.

- Maintaining Data Integrity: Automatically updating or adjusting related tables when a change occurs.

**Considerations:**

- Performance Impact: Triggers can affect database performance if not used judiciously, as they introduce additional processing for each relevant operation.
- Debugging Difficulty: Triggers can make debugging more challenging, as they execute automatically, which can lead to unexpected behavior if not carefully managed.
- Complexity: Overusing triggers can complicate database design and maintenance, making it harder to understand data flows.

```
create database trigger_practise;
use trigger_practise;

create table customers(id int auto_increment primary key,
name varchar(100),
email varchar(100));

create table email_changes_log(
id int auto_increment primary key,
customer_id int,
old_email varchar(100),
new_email varchar(100),
changed_at timestamp default current_timestamp);

insert into customers(name,email) values('Auahdahd','dqhduiqwh@gmail.com');

select * from customers;

DELIMITER //
CREATE TRIGGER log_email_changes
before update on customers
for each row
begin
     if old.email!=new.email then
          insert into email_changes_log(customer_id,old_email,new_email)
          values(old.id,old.email, new.email);
   end if;
end//

delimiter ;

select * from email_changes_log;
update customers set email='jdiejweiod@gmail.com' where id =1;
select * from customers;
```

# DAY 20 (14 October)

## Spring Boot Overview

Spring Boot is an extension of the Spring framework, designed to simplify the development of Java-based applications. It eliminates boilerplate configurations and provides embedded servers, allowing developers to quickly build stand-alone, production-ready applications.

## Key Concepts:

### 1. Inversion of Control (IoC)

IoC is a design principle where the control of object creation and dependency management is transferred from the application to the framework (Spring IoC container).
Instead of creating objects manually, the Spring container manages them, thus promoting loose coupling and better code management.

### 2. Beans

In Spring, a bean is an object managed by the Spring IoC container. Beans are created, configured, and managed by Spring.
Beans are defined either through annotations or XML configuration, and Spring ensures their lifecycle is managed.

### 3. Dependency Injection (DI)

DI is a design pattern used to inject dependencies into an object, rather than the object controlling its own dependencies.
Spring uses DI to manage object dependencies, making the code more testable and modular.
Types of DI:
Constructor Injection: Dependencies are passed through the constructor.
Setter Injection: Dependencies are set using setter methods.
Field Injection: Dependencies are directly injected into fields.

**4. @Autowired Annotation**
@Autowired is used to automatically inject beans (dependencies) by the Spring container.
It can be applied to constructors, setters, or fields.
Spring automatically resolves and injects the appropriate bean at runtime.

**5. getBean() Method**
The getBean() method is used to retrieve a specific bean from the Spring IoC container.
It can be used to programmatically fetch beans by name or type.

**6. @PostConstruct Annotation**
@PostConstruct is used to annotate a method that should be executed after the Spring container initializes the bean.
It's often used for initialization logic that should run once the bean's dependencies have been injected.

# # 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.

# DAY 21 (15 October)

My first Spring Boot Program :

```java
// HelloWorld.java
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class helloWorldApplication {
    public static void main(String[] args) {
        SpringApplication.run(helloWorldApplication.class, args);
        System.out.println("hello world from main class");
    }
}

// HelloWorldController.java
package com.example;

import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.GetMapping;

@RestController
public class HelloWorldController {
    @GetMapping("/hello")
    public String getMethodName() {
        return "Hello world";
    }
}
```

## Flow of a Spring Boot program :

**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.

```java
// File: WeatherApiApplication.java
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class WeatherApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(WeatherApiApplication.class, args);
    }
}

// File: WeatherConfig.java
package com.example;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

@Configuration
@ConfigurationProperties(prefix = "weather")
public class WeatherConfig {
    private String defaultCondition;

    public String getDefaultCondition() {
        return defaultCondition;
    }

    public void setDefaultCondition(String defaultCondition) {
        this.defaultCondition = defaultCondition;
    }
}

// File: WeatherController.java
package com.example;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import javax.validation.constraints.NotBlank;
import org.springframework.validation.annotation.Validated;

@RestController
@RequestMapping("/api")
@Validated
public class WeatherController {

    private final WeatherService weatherService;

    @Autowired
    public WeatherController(WeatherService weatherService) {
        this.weatherService = weatherService;
    }

    @GetMapping("/weather/{city}")
    public String getWeather(@PathVariable @NotBlank String city) {
        return weatherService.getWeather(city);
    }

    @PostMapping("/weather/{city}")
    public String updateWeather(@PathVariable @NotBlank String city,
                                @RequestParam @NotBlank String condition) {
        weatherService.updateWeather(city, condition);
        return "Weather updated for " + city;
    }
}

// File: WeatherService.java
package com.example;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.HashMap;
import java.util.Map;

@Service
public class WeatherService {

    private final WeatherConfig weatherConfig;
    private Map<String, String> weatherData = new HashMap<>();

    @Autowired
    public WeatherService(WeatherConfig weatherConfig) {
        this.weatherConfig = weatherConfig;
    }

    public String getWeather(String city) {
        return weatherData.getOrDefault(city, weatherConfig.getDefaultCondition());
    }

    public void updateWeather(String city, String condition) {
        weatherData.put(city, condition);
    }
}
```

**This code represents a simple Spring Boot application for managing weather conditions. Here's a brief description:**

1. **WeatherApiApplication**: The main class with `@SpringBootApplication` that starts the Spring Boot application.
2. **WeatherConfig**: A configuration class with `@Configuration` and `@ConfigurationProperties` annotations. It reads a property `weather.default-condition` from the `application.properties` file and provides getter and setter methods for the default weather condition.
3. **WeatherController**: A REST controller that handles HTTP requests. It has:
    - A `GET` endpoint `/api/weather/{city}` to return the weather for a specific city, or the default condition if no data exists.
    - A `POST` endpoint `/api/weather/{city}` to update the weather condition for a city.
4. **WeatherService**: A service class that contains the business logic. It:
    - Retrieves the weather for a city, falling back to the default condition if the city is not found.
    - Updates the weather condition for a city using an in-memory map (`weatherData`).

This structure uses Spring Boot's features for configuration, dependency injection, and RESTful web services.

# DAY 22 (16 October)

## Views in SQL :

```
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:
sqlCopy-- Create the employees table
CREATE TABLE employees (
    id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department VARCHAR(50),
    salary DECIMAL(10, 2)
);

-- Insert some sample data
INSERT INTO employees VALUES
(1, 'John', 'Doe', 'IT', 75000),
(2, 'Jane', 'Smith', 'HR', 65000),
(3, 'Mike', 'Johnson', 'IT', 80000),
(4, 'Emily', 'Brown', 'Finance', 70000);
```

## # 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.

Remember, unlike Oracle's v$session, which provides a wealth of session-specific information, MySQL's tools focus more on the query itself and basic session data. If you need more detailed session information, you might need to explore additional monitoring tools or plugins.

# Spring boot (Weather Application) :

```java
// File: WeatherConfig.java
package com.example.weathor;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
import java.util.Map;
import java.util.HashMap;

@Configuration
@ConfigurationProperties(prefix = "weather")
public class WeatherConfig {
    private String defaultCondition;
    private Map<String, TemperatureRange> cityTemperatures = new HashMap<>();

    // Getters and setters
    public String getDefaultCondition() { return defaultCondition; }
    public void setDefaultCondition(String defaultCondition) { this.defaultCondition = defaultCondition; }
    public Map<String, TemperatureRange> getCityTemperatures() { return cityTemperatures; }
    public void setCityTemperatures(Map<String, TemperatureRange> cityTemperatures) { this.cityTemperatures = cityTemperatures; }

    public static class TemperatureRange {
        private int min;
        private int max;

        // Getters and setters
        public int getMin() { return min; }
        public void setMin(int min) { this.min = min; }
        public int getMax() { return max; }
        public void setMax(int max) { this.max = max; }
    }
}

// File: WeatherRecord.java
package com.example;

import org.springframework.stereotype.Component;
import jakarta.persistence.*;

@Entity
@Table(name="weather_records")
@Component
public class WeatherRecord {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false,unique = true)
    private String city;
    @Column(nullable = false)
    private int temperature;
    @Column(nullable = false)
    private String condition;

    public WeatherRecord(){

    }

    public void initialize(String city, int temperature){
        this.city = city;
        this.temperature = temperature;
    }

    public String getCity(){
        return city;
    }

    public String getCondition(){
        return condition;
    }
    public void setCity(String city){
        this.city = city;
    }
    public void setCondition(String condition){
        this.condition = condition;
    }
    public void setTemperature(int temperature){
        this.temperature = temperature;
    }
    public int getTemperature(){
        return temperature;
    }
}
```

```java
// File: WeatherService.java
package com.example.weather;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Service;
import java.util.HashMap;
import java.util.Map;

@Service
public class WeatherService {
    private final WeatherConfig weatherConfig;
    private final ApplicationContext context;
    private final Map<String, WeatherRecord> weatherData = new HashMap<>();

    @Autowired
    public WeatherService(WeatherConfig weatherConfig, ApplicationContext context) {
        this.weatherConfig = weatherConfig;
        this.context = context;
    }

    public String getWeather(String city) {
        WeatherRecord record = weatherData.get(city);
        if (record == null) {
            return weatherConfig.getDefaultCondition();
        }
        return String.format("The temperature in %s is %d°C. Condition: %s",
                             city, record.getTemperature(), record.getCondition());
    }

    public void updateWeather(String city, int temperature) {
        WeatherRecord record = weatherData.computeIfAbsent(city, k -> {
            WeatherRecord newRecord = context.getBean(WeatherRecord.class);
            newRecord.initialize(city, temperature);
            return newRecord;
        });
        record.setTemperature(temperature);
        record.setCondition(determineCondition(city, temperature));
    }

    private String determineCondition(String city, int temperature) {
        WeatherConfig.TemperatureRange range = weatherConfig.getCityTemperatures().get(city);
        if (range == null) {
            return weatherConfig.getDefaultCondition();
        }
        if (temperature < range.getMin()) {
            return "Cold";
        } else if (temperature > range.getMax()) {
            return "Hot";
        } else {
            return "Pleasant";
        }
    }

    public Map<String, String> getAllWeatherData() {
        Map<String, String> result = new HashMap<>();
        weatherData.forEach((city, record) ->
            result.put(city, String.format("%d°C, %s", record.getTemperature(), record.getCondition()))
        );
        return result;
    }
}
```

```java
// File: WeatherController.java
package com.example.weather;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.Map;

@RestController
@RequestMapping("/api/weather")
public class WeatherController {
    private final WeatherService weatherService;

    @Autowired
    public WeatherController(WeatherService weatherService) {
        this.weatherService = weatherService;
    }

    @GetMapping("/{city}")
    public String getWeather(@PathVariable String city) {
        return weatherService.getWeather(city);
    }

    @PostMapping("/{city}")
    public String updateWeather(@PathVariable String city, @RequestParam int temperature) {
        weatherService.updateWeather(city, temperature);
        return "Weather updated for " + city;
    }

    @GetMapping
    public Map<String, String> getAllWeather() {
        return weatherService.getAllWeatherData();
    }
}

// File: application.properties
weather.default-condition=Moderate
weather.city-temperatures.new-york.min=10
weather.city-temperatures.new-york.max=25
weather.city-temperatures.london.min=5
weather.city-temperatures.london.max=20
weather.city-temperatures.tokyo.min=15
weather.city-temperatures.tokyo.max=30
```

## Changes made in previous application :

1. **City-Specific Temperature Ranges:**
    - In the `WeatherConfig` class, a new property `cityTemperatures` (a map of cities to their

`TemperatureRange` values) was added. This stores min/max temperature ranges for each city.
    - This allows each city to have a different weather condition based on its temperature range, rather than all cities defaulting to the same condition.
2. **How it works:**
    - Each city's temperature is compared to its specific range, and the condition (e.g., "Cold," "Hot," "Pleasant") is determined accordingly. If no specific city range is provided, it falls back to the default condition.
3. **WeatherRecord Entity:**
    - A new `WeatherRecord` class was added to represent weather records for each city, including `city`, `temperature`, and `condition`.
    - This class is used to store the weather information for each city in-memory, and it can later be expanded to persist data to a database if needed (since it's annotated with `@Entity`).
4. **How it works:**
    - A `WeatherRecord` object is created for each city and stored in the `weatherData` map within the `WeatherService`. Each record includes city-specific weather data (temperature and condition).
5. **Improved `WeatherService` Logic:**
    - The service now checks if a `WeatherRecord` exists for a city. If not, it creates a new record using the `WeatherConfig` data and stores it in-memory.
    - The `updateWeather()` method was modified to update the temperature and automatically determine the condition based on the city's temperature range from `WeatherConfig`.
6. **How it works:**
    - When updating a city's weather, the method now calculates the condition dynamically based on the temperature and predefined ranges (e.g., if the temperature exceeds the maximum, the

condition becomes "Hot"). The condition is updated based on the city's specific configuration, not a global default.

7. **Determine Condition Method:**
    - The new `determineCondition()` method calculates the weather condition based on the city's temperature. It uses the `TemperatureRange` for that city from `WeatherConfig` to decide whether the weather is "Cold," "Pleasant," or "Hot."
8. **Updated Application Properties:**
    - The `application.properties` file now includes temperature ranges for different cities (e.g., New York, London, Tokyo), allowing the conditions for these cities to be calculated dynamically.

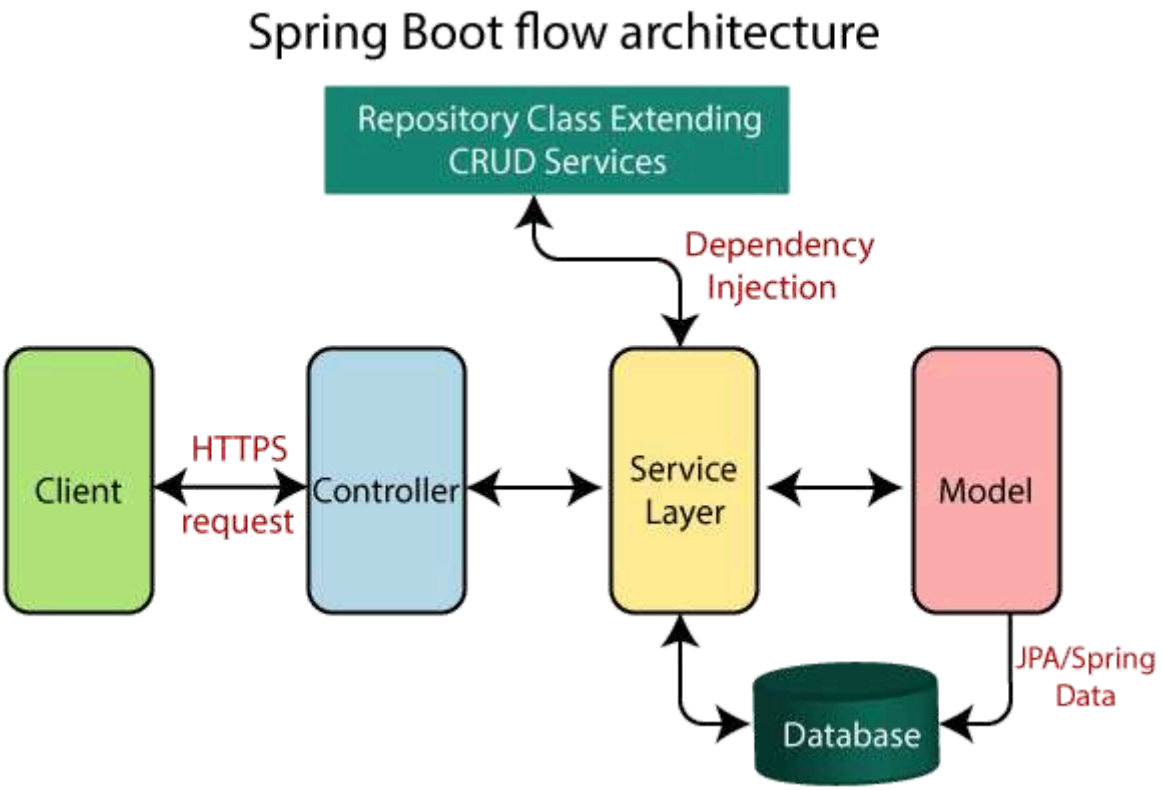
---

## How It Works Now:

- Dynamic Weather Per City: Each city now has specific temperature ranges defined in `application.properties`. When a request is made to get or update the weather for a city, the system checks these ranges to determine if the condition is "Cold," "Pleasant," or "Hot."
- Temperature-Based Logic: The service now dynamically assigns weather conditions based on the temperature for each city, instead of returning the same default condition for all cities.
- Weather Records: Each city has its own `WeatherRecord` that stores city-specific weather data (city name, temperature, and condition), improving the accuracy of weather data handling across cities.

This solves the original issue of returning the same weather for all cities by adding city-specific logic and temperature ranges.

# DAY 23 (17 October)




## Explanation of Each Component

1. **WeatherController:**
    - Handles HTTP requests and routes them to the service layer.
    - Methods:
        - `getAllWeather()`: Retrieves all weather data.
        - `getWeatherByCity(city)`: Fetches weather data for a specific city.
        - `addWeather(weatherRecord)`: Adds a new weather entry.
        - `updateWeather(id, weatherRecord)`: Updates weather for a given city using the ID.
        - `deleteWeather(id)`: Deletes weather data by ID.

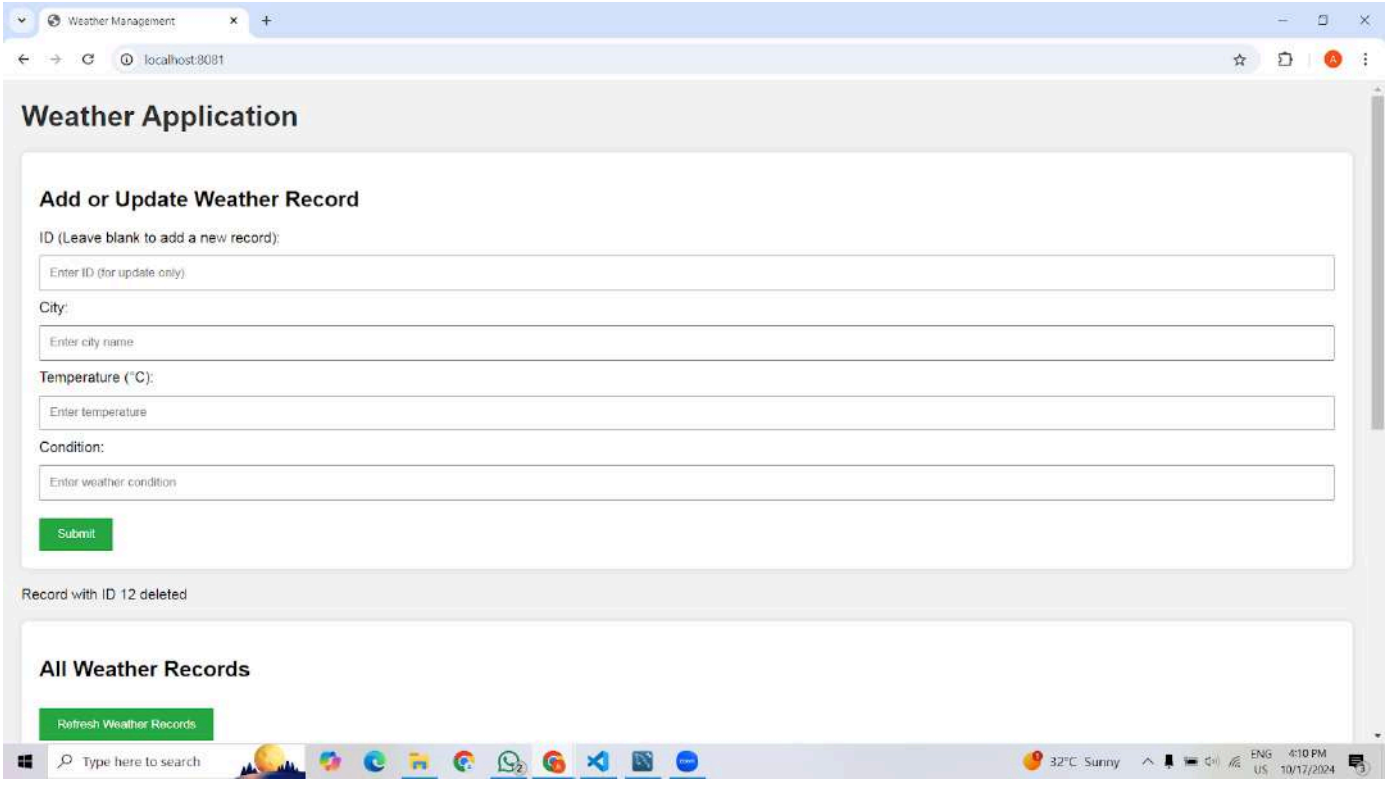
    - Uses `ResponseEntity` to return proper HTTP status codes like `404 Not Found` and `200 OK`.
2. **WeatherService:**
    - Processes the business logic of the application.
    - Methods:
        - `getAllWeather()`: Fetches all weather records from the database.
        - `getWeatherByCity(city)`: Retrieves weather by city, throws an error if not found.
        - `addWeather(weatherRecord)`: Adds a new weather record to the database.
        - `updateWeather(weatherRecord)`: Updates an existing weather record.
        - `deleteWeather(id)`: Deletes a weather record by ID.
3. **WeatherConfig:**
    - Loads weather-related configurations (default condition, temperature ranges) from `application.properties`.
    - Stores temperature ranges for different cities using a `HashMap`.
4. **WeatherRecord:**
    - Entity representing the `weather_records` table in the database.
    - Columns:
        - `id`: Unique identifier.
        - `city`: Name of the city (unique).
        - `temperature`: City's temperature.
        - `weatherCondition`: Current weather condition.
5. **WeatherRepository:**
    - A JPA repository interface that allows CRUD operations on the `WeatherRecord` entity.
    - Methods:
        - `findByCity(city)`: Retrieves weather data based on the city.

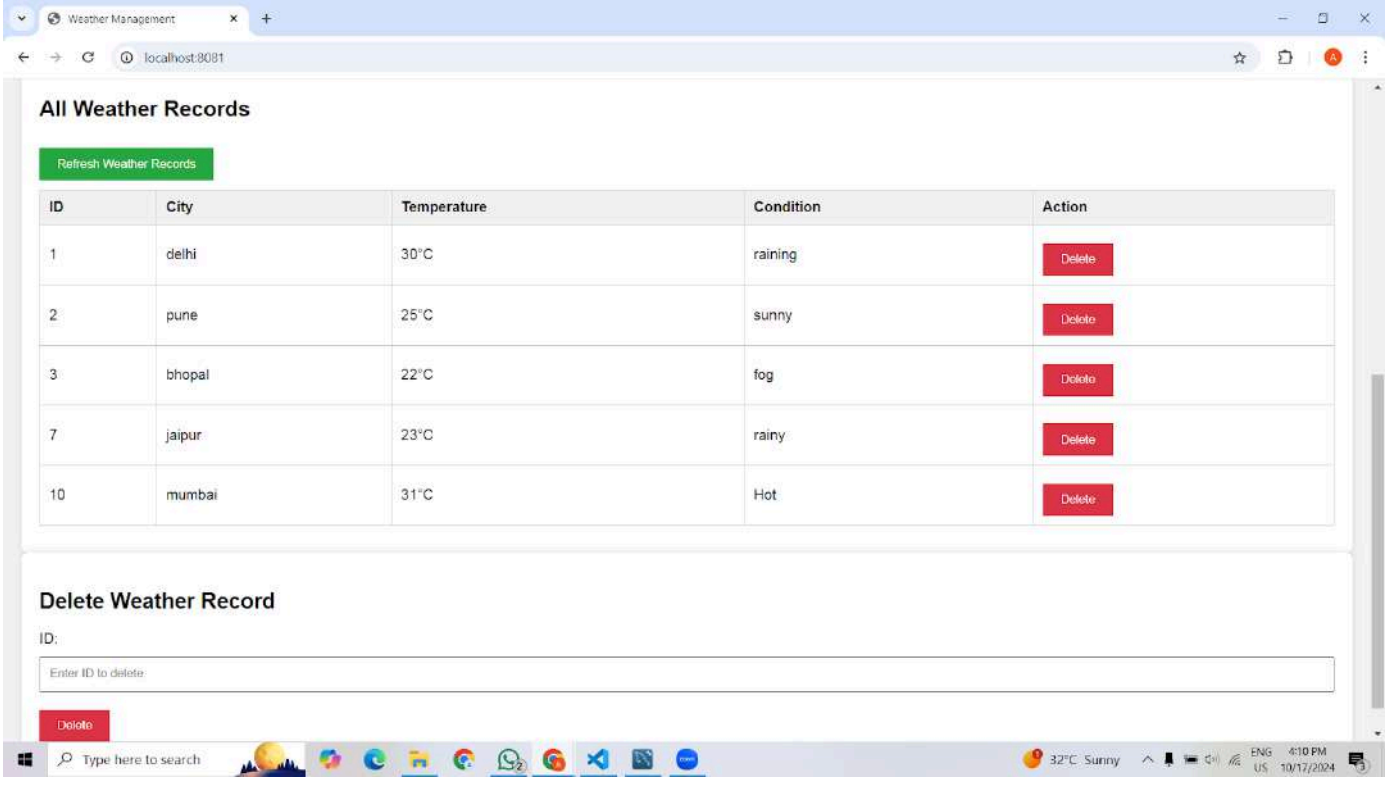
**How Changes Work:**

- Old Issue: All cities returned the same default weather condition because the weather service didn't have city-specific records.
- New Changes:
  - Introduced `WeatherRecord` as an entity to store weather data in the database.
  - Controller now interacts with the database via `WeatherService`.
  - Repository added to allow database operations.
  - Temperature Range logic in `WeatherConfig` ensures correct conditions are shown based on city-specific temperatures.

```java
package com.example;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;
@Repository
public interface WeatherRepositry extends JpaRepository<WeatherRecord,Long> {

    Optional<WeatherRecord> findByCity(String city);
}
```

# WEATHER APPLICATION :



# DAY 24 (18 October)

# SQL PRACTICE :

```sql
-- 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 daily 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.

CREATE TABLE employee_sales (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(50),
    department VARCHAR(50),
    sales_amount DECIMAL(10, 2),
    sales_date DATE
);

-- Sample data (insert statements) would go here

INSERT INTO employee_sales (employee_id, employee_name, department, sales_amount, sales_date) VALUES
(1, 'John Doe', 'Electronics', 1500.00, '2023-01-15'),
(2, 'Jane Smith', 'Clothing', 2000.00, '2023-01-16'),
(3, 'Mike Johnson', 'Electronics', 1800.00, '2023-01-17'),
(4, 'Emily Brown', 'Home Goods', 1200.00, '2023-01-18'),
(5, 'David Lee', 'Clothing', 2200.00, '2023-01-19'),
(6, 'Sarah Wilson', 'Electronics', 1600.00, '2023-01-20'),
(7, 'Tom Harris', 'Home Goods', 1300.00, '2023-01-21'),
(8, 'Lisa Chen', 'Clothing', 1900.00, '2023-01-22');

WITH DepartmentSales AS (
    SELECT
  department,
  SUM(sales_amount) AS total_sales,
  AVG(sales_amount) AS avg_daily_sales,
  COUNT(DISTINCT employee_id) AS num_employees
    FROM employee_sales
    GROUP BY department
)
SELECT
department,
total_sales,
avg_daily_sales,
num_employees,
RANK() OVER (ORDER BY total_sales DESC) AS department_rank
FROM DepartmentSales
ORDER BY total_sales DESC;
```

```sql
-- 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:

-- Ranks employees within each department based on the number of projects they've completed.
-- Calculates the average project duration for each department.
-- Identifies projects that took longer than the department's average.
-- Finds the top 3 most efficient employees in each department (based on average project duration).
-- Compares each project's duration to the previous project in the same department.

CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50)
);

CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(50),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);

CREATE TABLE Projects (
    ProjectID INT PRIMARY KEY,
    ProjectName VARCHAR(100),
    DepartmentID INT,
    StartDate DATE,
    EndDate DATE,
    EmployeeID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID),
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);

WITH ProjectStats AS (
    -- Calculate project duration and count
    SELECT
        p.ProjectID,
        p.ProjectName,
        p.DepartmentID,
        p.EmployeeID,
        e.EmployeeName,
        d.DepartmentName,
        DATEDIFF(day, p.StartDate, p.EndDate) AS ProjectDuration,
        COUNT(*) OVER (PARTITION BY p.EmployeeID) AS ProjectCount,
        AVG(DATEDIFF(day, p.StartDate, p.EndDate)) OVER (PARTITION BY p.DepartmentID) AS AvgDeptDuration
    FROM
        Projects p
        JOIN Employees e ON p.EmployeeID = e.EmployeeID
        JOIN Departments d ON p.DepartmentID = d.DepartmentID
),
RankedEmployees AS (
    -- Rank employees within departments based on project count
    SELECT
        *,
        RANK() OVER (PARTITION BY DepartmentID ORDER BY ProjectCount DESC) AS EmployeeRank,
        DENSE_RANK() OVER (PARTITION BY DepartmentID ORDER BY ProjectDuration) AS EfficiencyRank
    FROM
        ProjectStats
),
ProjectComparison AS (
    -- Compare project duration to previous project in the same department
    SELECT
        *,
        LAG(ProjectDuration) OVER (PARTITION BY DepartmentID ORDER BY StartDate) AS PrevProjectDuration,
        ProjectDuration - LAG(ProjectDuration) OVER (PARTITION BY DepartmentID ORDER BY StartDate) AS DurationDifference
    FROM
        Projects
)
SELECT
    re.DepartmentName,
    re.EmployeeName,
    re.ProjectCount,
    re.EmployeeRank,
    re.EfficiencyRank,
    p.ProjectName,
    p.ProjectDuration,
    p.AvgDeptDuration,
    CASE
        WHEN p.ProjectDuration > p.AvgDeptDuration THEN 'Above Average'
        WHEN p.ProjectDuration < p.AvgDeptDuration THEN 'Below Average'
        ELSE 'Average'
    END AS DurationComparison,
    pc.PrevProjectDuration,
    pc.DurationDifference
FROM
    RankedEmployees re
    JOIN ProjectStats p ON re.ProjectID = p.ProjectID
    JOIN ProjectComparison pc ON p.ProjectID = pc.ProjectID
WHERE
    re.EfficiencyRank <= 3  -- Top 3 most efficient employees
ORDER BY
    re.DepartmentName,
    re.EfficiencyRank,
    p.ProjectDuration;
```

# DAY 25 (21 October)

```java
package com.example;


import java.util.List;


import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.OneToMany;
import jakarta.persistence.Table;


/**
 * This class represents a Restaurant entity in the database.
 * It contains the restaurant's id, name, cuisine, and a list of reviews.
 */
@Entity
@Table(name = "restaurant")
public class Restaurant {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String name;
    @Column(nullable = false)
    private String cusine;
    @OneToMany(mappedBy = "restaurant", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Review> reviews;


    //getters and setters
    public Long getId(){
        return id;
    }
    public void setId(Long id){
        this.id = id;
    }
    public String getName(){
        return name;
    }
    public void setName(String name){
        this.name = name;
    }
    public String getCusine(){
        return cusine;
    }
    public void setCusine(String cusine){
        this.cusine = cusine;
    }
    public List<Review> getReviews(){
        return reviews;
    }
    public void setReviews(List<Review> reviews){
        this.reviews = reviews;
    }

}
```

```java
package com.example;


import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;


/**
 * This class represents a Review entity in the database.
 * It contains the review's id, reviewer's name, rating, associated restaurant, and the comment.
 */
@Entity
@Table(name = "review")
public class Review {
    /**
     * The unique identifier for the review.
     */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    /**
     * The name of the reviewer. This field cannot be null.
     */
    @Column(nullable = false)
    private String reviewerName;
    /**
     * The rating given by the reviewer. This field cannot be null.
     */
    @Column(nullable = false)
    private int rating;
    /**
     * The restaurant associated with this review. This field cannot be null.
     */
    @ManyToOne
    @JoinColumn(name = "restaurant_id", nullable = false)
    private Restaurant restaurant;
    /**
     * The comment left by the reviewer. This field cannot be null.
     */
    @Column(nullable = false)
    private String comment;


    //getters and setters
    /**
     * Returns the unique identifier for the review.
     * @return the review's id
     */
    public Long getId(){
        return id;
    }
    /**
     * Sets the unique identifier for the review.
     * @param id the review's id
     */
```

```java
public void setId(Long id){
        this.id = id;
    }
    /**
     * Returns the name of the reviewer.
     * @return the reviewer's name
     */
    public String getReviewerName(){
        return reviewerName;
    }
    /**
     * Sets the name of the reviewer.
     * @param reviewerName the reviewer's name
     */
    public void setReviewerName(String reviewerName){
        this.reviewerName = reviewerName;
    }
    /**
     * Returns the rating given by the reviewer.
     * @return the rating
     */
    public int getRating(){
        return rating;
    }
    /**
     * Sets the rating given by the reviewer.
     * @param rating the rating
     */
    public void setRating(int rating){
        this.rating = rating;
    }
    /**
     * Returns the restaurant associated with this review.
     * @return the associated restaurant
     */
    public Restaurant getRestaurant(){
        return restaurant;
    }
    /**
     * Sets the restaurant associated with this review.
     * @param restaurant the associated restaurant
     */
    public void setRestaurant(Restaurant restaurant){
        this.restaurant = restaurant;
    }
    /**
     * Returns the comment left by the reviewer.
     * @return the comment
     */
    public String getComment(){
        return comment;
    }
    /**
     * Sets the comment left by the reviewer.
     * @param comment the comment
     */
    public void setComment(String comment){
        this.comment = comment;
    }

}
```

```java
package com.example;


import java.util.List;
import java.util.Optional;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;


/**
 * This class is a Spring Boot REST controller that handles HTTP requests related to restaurants.
 * It provides endpoints for creating a new restaurant, retrieving all restaurants, getting a specific restaurant by ID,
 * adding a review to a restaurant, and retrieving all reviews for a restaurant.
 */
@RestController
@RequestMapping("/api/restaurants")
public class RestaurantController {
    @Autowired
    private RestaurantRepository restaurantRepository; // Injects the RestaurantRepository for database operations on restaurants.
    @Autowired
    private ReviewRepository reviewRepository; // Injects the ReviewRepository for database operations on reviews.


    /**
     * Creates a new restaurant and saves it to the database.
     *
     * @param restaurant The restaurant object to be saved.
     * @return A ResponseEntity containing the saved restaurant and a HTTP status of CREATED.
     */
    @PostMapping
    public ResponseEntity<Restaurant> addRestaurant(@RequestBody Restaurant restaurant) {
        Restaurant savedRestaurant = restaurantRepository.save(restaurant); // Saves the restaurant to the database.
        return new ResponseEntity<>(savedRestaurant, HttpStatus.CREATED); // Returns the saved restaurant with a HTTP status of CREATED.
    }


    /**
     * Retrieves all restaurants from the database.
     *
     * @return A ResponseEntity containing a list of all restaurants and a HTTP status of OK.
     */
    @GetMapping
    public ResponseEntity<List<Restaurant>> getAllRestaurants() {
        List<Restaurant> restaurants = restaurantRepository.findAll(); // Retrieves all restaurants from the database.
        return new ResponseEntity<>(restaurants, HttpStatus.OK); // Returns the list of restaurants with a HTTP status of OK.
    }
```

```java
/**
     * Retrieves a specific restaurant by its ID from the database.
     *
     * @param id The ID of the restaurant to be retrieved.
     * @return A ResponseEntity containing the restaurant if found, otherwise a HTTP status of NOT_FOUND.
     */
    @GetMapping("/{id}")
    public ResponseEntity<Restaurant> getRestaurantById(@PathVariable Long id) {
        Optional<Restaurant> restaurant = restaurantRepository.findById(id); // Tries to find the restaurant by its ID.
        if (restaurant.isPresent()) {
            return new ResponseEntity<>(restaurant.get(), HttpStatus.OK); // Returns the restaurant if found with a HTTP status of OK.
        }
        return new ResponseEntity<>(HttpStatus.NOT_FOUND); // Returns a HTTP status of NOT_FOUND if the restaurant is not found.
    }


    /**
     * Adds a review to a specific restaurant.
     *
     * @param id The ID of the restaurant to which the review is being added.
     * @param review The review object to be added.
     * @return A ResponseEntity containing the saved review and a HTTP status of CREATED if the restaurant exists, otherwise a HTTP status of NOT_FOUND.
     */
    @PostMapping("/{id}/reviews")
    public ResponseEntity<Review> addReview(@PathVariable Long id, @RequestBody Review review) {
        Optional<Restaurant> restaurantOpt = restaurantRepository.findById(id); // Tries to find the restaurant by its ID.
        if (restaurantOpt.isPresent()) {
            Restaurant restaurant = restaurantOpt.get(); // Gets the restaurant if found.
            review.setRestaurant(restaurant); // Sets the restaurant for the review.
            Review savedReview = reviewRepository.save(review); // Saves the review to the database.
            return new ResponseEntity<>(savedReview, HttpStatus.CREATED); // Returns the saved review with a HTTP status of CREATED.
        }
        return new ResponseEntity<>(HttpStatus.NOT_FOUND); // Returns a HTTP status of NOT_FOUND if the restaurant is not found.
    }


    /**
     * Retrieves all reviews for a specific restaurant.
     *
     * @param id The ID of the restaurant for which the reviews are being retrieved.
     * @return A ResponseEntity containing a list of reviews if the restaurant exists, otherwise a HTTP status of NOT_FOUND.
     */
    @GetMapping("/{id}/reviews")
    public ResponseEntity<List<Review>> getReviewsForRestaurant(@PathVariable Long id) {
        Optional<Restaurant> restaurantOpt = restaurantRepository.findById(id); // Tries to find the restaurant by its ID.
        if (restaurantOpt.isPresent()) {
            Restaurant restaurant = restaurantOpt.get(); // Gets the restaurant if found.
            List<Review> reviews = restaurant.getReviews(); // Gets the list of reviews for the restaurant.
            return new ResponseEntity<>(reviews, HttpStatus.OK); // Returns the list of reviews with a HTTP status of OK.
        }
        return new ResponseEntity<>(HttpStatus.NOT_FOUND); // Returns a HTTP status of NOT_FOUND if the restaurant is not found.
    }
}
```

```java
package com.example;


import java.util.List;
import java.util.Optional;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;


/**
 * This class provides service methods for managing restaurants and their reviews.
 * It utilizes Spring Data JPA repositories for database operations.
 */
@Service
public class RestaurantService {

    @Autowired
    private RestaurantRepository restaurantRepository; // Injects the RestaurantRepository for database operations on restaurants.


    @Autowired
    private ReviewRepository reviewRepository; // Injects the ReviewRepository for database operations on reviews.


    /**
     * Adds a new restaurant to the database.
     *
     * @param restaurant The restaurant object to be saved.
     * @return The saved restaurant object.
     */
    public Restaurant addRestaurant(Restaurant restaurant) {
        return restaurantRepository.save(restaurant); // Saves the restaurant to the database and returns the saved object.
    }


    /**
     * Retrieves all restaurants from the database.
     *
     * @return A list of all restaurants in the database.
     */
    public List<Restaurant> getAllRestaurants() {
        return restaurantRepository.findAll(); // Retrieves all restaurants from the database and returns them as a list.
    }


    /**
     * Retrieves a specific restaurant by its ID from the database.
     *
     * @param id The ID of the restaurant to be retrieved.
     * @return An Optional containing the restaurant if found, otherwise an empty Optional.
     */
    public Optional<Restaurant> getRestaurantById(Long id) {
        return restaurantRepository.findById(id); // Tries to find the restaurant by its ID and returns an Optional containing the result.
    }


    /**
     * Adds a review to a specific restaurant.
     *
     * @param restaurantId The ID of the restaurant to which the review is being added.
     * @param review The review object to be added.
     * @return The saved review object if the restaurant exists, otherwise null.
     */
    public Review addReview(Long restaurantId, Review review) {
        Optional<Restaurant> restaurantOpt = restaurantRepository.findById(restaurantId); // Tries to find the restaurant by its ID.
        if (restaurantOpt.isPresent()) {
            Restaurant restaurant = restaurantOpt.get(); // Gets the restaurant if found.
            review.setRestaurant(restaurant); // Sets the restaurant for the review.
            return reviewRepository.save(review); // Saves the review to the database and returns the saved object.
        }
        return null; // Returns null if the restaurant is not found.
    }


    /**
     * Retrieves all reviews for a specific restaurant.
     *
     * @param restaurantId The ID of the restaurant for which the reviews are being retrieved.
     * @return A list of reviews if the restaurant exists, otherwise null.
     */
    public List<Review> getReviewsForRestaurant(Long restaurantId) {
        Optional<Restaurant> restaurantOpt = restaurantRepository.findById(restaurantId); // Tries to find the restaurant by its ID.
        if (restaurantOpt.isPresent()) {
            return restaurantOpt.get().getReviews(); // Gets the list of reviews for the restaurant if found.
        }
        return null; // Returns null if the restaurant is not found.
    }
}
```

```java
package com.example;


import java.util.List;


import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;


@Repository
public interface RestaurantRepository extends JpaRepository<Restaurant, Long> {



    List<Restaurant> findByCusine(String cusine);


    @Query("SELECT r FROM Restaurant r WHERE r.name LIKE %?1%")
    List<Restaurant> searchByName(String name);


    @Query("SELECT r FROM Restaurant r WHERE r.cusine LIKE %?1%")
    List<Restaurant> searchByCusine(String cusine);


    @Query("select r from Restaurant r join r.reviews rv group by r having avg(rv.rating) > ?1")
    List<Restaurant> searchByRating(double rating);
}
```

```java
package com.example;


import java.util.List;
import java.util.Optional;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;


@Service
public class ReviewService {


    @Autowired
    private ReviewRepository reviewRepository;


    public List<Review> getAllReviews() {
        return reviewRepository.findAll();
    }


    public Optional<Review> getReviewById(Long id) {
        return reviewRepository.findById(id);
    }


    public Review saveReview(Review review) {
        return reviewRepository.save(review);
    }


    public void deleteReview(Long id) {
        reviewRepository.deleteById(id);
    }
}
```

```java
package com.example;


import java.util.List;
import java.util.Optional;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;


@RestController
@RequestMapping("/api/reviews")
public class ReviewController {


    @Autowired
    private ReviewService reviewService;


    @GetMapping
    public List<Review> getAllReviews() {
        return reviewService.getAllReviews();
    }


    @GetMapping("/{id}")
    public ResponseEntity<Review> getReviewById(@PathVariable Long id) {
        Optional<Review> review = reviewService.getReviewById(id);
        return review.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());
    }


    @PostMapping
    public Review createReview(@RequestBody Review review) {
        return reviewService.saveReview(review);
    }


    @PutMapping("/{id}")
    public ResponseEntity<Review> updateReview(@PathVariable Long id, @RequestBody Review reviewDetails) {
        Optional<Review> review = reviewService.getReviewById(id);
        if (review.isPresent()) {
            reviewDetails.setId(id);
            return ResponseEntity.ok(reviewService.saveReview(reviewDetails));
        }
        return ResponseEntity.notFound().build();
    }


    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteReview(@PathVariable Long id) {
        reviewService.deleteReview(id);
        return ResponseEntity.noContent().build();
    }
}
```

# DAY 26 (22 October)

**Scope:**

scope defines the lifecycle and visibility of a bean in the Spring container. By default, beans in Spring are `singleton`, but Spring provides several other scopes that control how and when beans are created and managed.

Common Bean Scopes in Spring:

1. Singleton (default)
2. Prototype
3. Request
4. Session
5. Application
6. WebSocket

**Singleton (Default)**

- **Definition**: Spring creates only one instance of the bean per Spring IoC container. All requests for that bean will return the same instance.
- **Usage**: This is the default scope in Spring and is used when you want a single shared instance of the bean across your application.
- **Lifecycle**: The bean is created when the Spring container starts and is destroyed when the container is shut down.

**Prototype**

- **Definition**: A new bean instance is created every time it is requested from the Spring container.

- **Usage**: Use this scope when you need a new instance of a bean each time it is injected or looked up.
- **Lifecycle**: The container creates a new instance every time the bean is requested. It does not manage the lifecycle of the bean beyond creation (e.g., it does not handle destruction).

**Request (Web Application Scope)**

- **Definition**: A single bean instance is created for each HTTP request. This scope is only valid in web-aware Spring applications.
- **Usage**: Use this scope when you want a new bean instance for each HTTP request. Useful for controllers and request-specific beans.
- **Lifecycle**: The bean is created at the beginning of an HTTP request and destroyed at the end of the request.

**Session (Web Application Scope)**

- **Definition**: A single bean instance is created for each HTTP session. This scope is also valid in web-aware applications.
- **Usage**: Use this scope for beans that should maintain session-specific data.
- **Lifecycle**: The bean is tied to an HTTP session, created when the session starts and destroyed when the session ends.

```java
package com.example;


import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;


@Entity
@Table(name="weather_records")
public class WeatherRecord {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false,unique = true)
    private String city;
    @Column(nullable = false)
    private int temperature;
    @Column(nullable = false)
    private String weatherCondition;


    public WeatherRecord(){


    }


    public void intialize(String city, int temperature){
        this.city = city;
        this.temperature = temperature;
    }


    public Long getId(){
        return id;
    }
    public void setId(Long id){
        this.id = id;
    }


    public String getCity(){
        return city;
    }


    public String getWeatherCondition(){
        return weatherCondition;
    }
    public void setCity(String city){
        this.city = city;
    }
    public void setWeatherCondition(String weatherCondition){
        this.weatherCondition = weatherCondition;
    }
    public void setTemperature(int temperature){
        this.temperature = temperature;
    }
    public int getTemperature(){
        return temperature;
    }
}
```

```java
package com.example;


import jakarta.validation.constraints.Max;
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Pattern;
import jakarta.validation.constraints.Size;


public class WeatherRequestDTO{
    @NotBlank(message = "City is required and cannot be blank")
    @Size(min= 3, max = 20, message = "City must be between 3 to 20 characters")
    private String city;

    @Min(value = -50, message = "temperature cannot be less than -50")
    @Max(value = 50, message = "temperature cannot be more than 50")
    private int temperature;
    @NotBlank(message = "weather is required and cannot be blank")
    @Size(min= 3, max = 20, message = "weather condition must be between 3 to 20 characters")
    @Pattern(regexp = "^(Sunny|cloudy|rainy|snowy)$", message = "weather condition must be sunny")

    private String weatherCondition;


    public String getCity() {
        return city;
    }


    public void setCity(String city) {
        this.city = city;
    }


    public int getTemperature() {
        return temperature;
    }


    public void setTemperature(int temperature) {
        this.temperature = temperature;
    }


    public String getWeatherCondition() {
        return weatherCondition;
    }


    public void setWeatherCondition(String weatherCondition) {
        this.weatherCondition = weatherCondition;
    }
}
```

```java
package com.example;


public class weatherValidationConstant {

    // City constraints
    public static final int CITY_MIN_LENGTH = 3;
    public static final int CITY_MAX_LENGTH = 20;

    // Temperature constraints
    public static final int TEMPERATURE_MIN = -50;
    public static final int TEMPERATURE_MAX = 50;

    // Weather condition constraints
    public static final int WEATHER_CONDITION_MIN_LENGTH = 3;
    public static final int WEATHER_CONDITION_MAX_LENGTH = 20;
    public static final String WEATHER_CONDITION_PATTERN = "^(Sunny|cloudy|rainy|snowy)$";
}
```

```java
package com.example;


import java.util.ArrayList;
import java.util.List;


import org.springframework.stereotype.Component;


@Component
public class WeatherValidator {

    public void validateWeatherCondition(WeatherRequestDTO weatherRequestDTO)
    {
        List<String> violations = new ArrayList<>();
        if(weatherRequestDTO.getTemperature() > 30 && "Snowy".equals(weatherRequestDTO.getWeatherCondition()))
        violations.add("snowy weather is not allowed when temperature is above 30 degrees");


        if(weatherRequestDTO.getTemperature() < 0 && "Sunny".equals(weatherRequestDTO.getWeatherCondition()))
        violations.add("snowy weather is not allowed when temperature is below 0 degrees");

        if(violations.isEmpty())
        {
            return;
        }
        throw new WeatherValidationException(String.join(" ",violations),violations);
    }
}
```

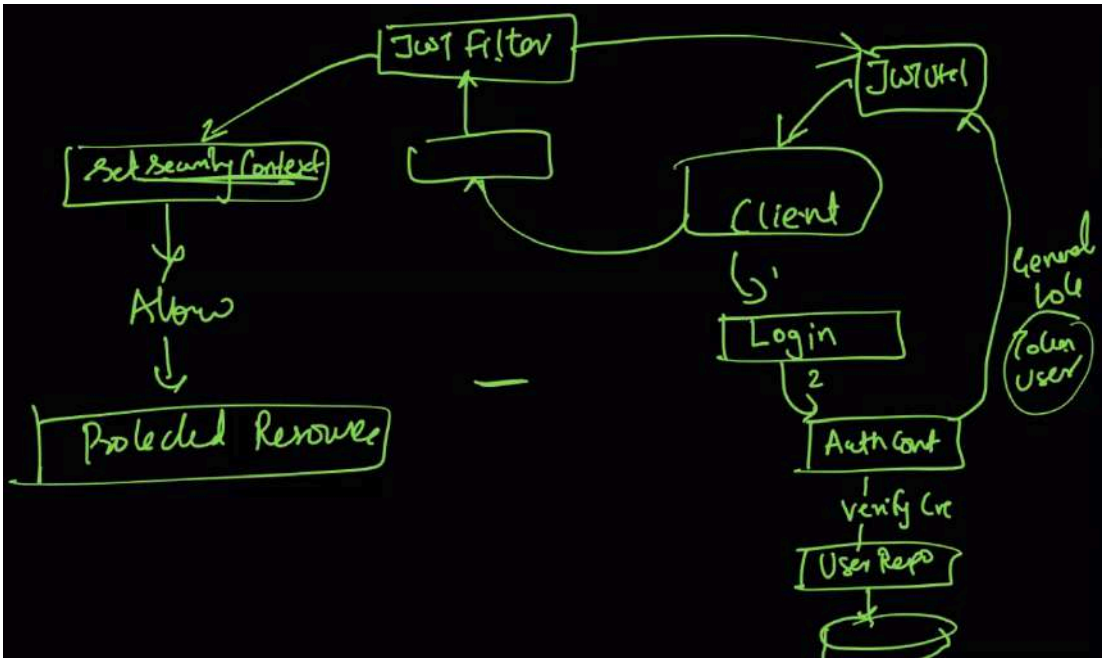
```java
package com.example.weather.controller;


import java.util.HashMap;
import java.util.Map;


import org.springframework.context.ApplicationContext;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;


import com.example.WeatherRecord;
import com.example.weather.service.UserWeatherPreferences;
import com.example.weather.service.WeatherQuery;
import com.example.weather.service.WeatherRequestLogger;
import com.example.weather.service.WeatherService;


@RestController
@RequestMapping("/weather")
public class WeatherController {
    private final WeatherService weatherService;              // Singleton
    private final WeatherRequestLogger requestLogger;         // Request Scope
    private final UserWeatherPreferences userPreferences;     // Session Scope
    private final ApplicationContext applicationContext;      // For Prototype


    public WeatherController(
            WeatherService weatherService,
            WeatherRequestLogger requestLogger,
            UserWeatherPreferences userPreferences,
            ApplicationContext applicationContext) {
        this.weatherService = weatherService;
        this.requestLogger = requestLogger;
        this.userPreferences = userPreferences;
        this.applicationContext = applicationContext;
    }


    @GetMapping("/{city}")
    public Map<String, Object> getWeather(@PathVariable String city) {
        // Request scope - log the request
        requestLogger.logRequest(city);


        // Session scope - add to recent searches
        userPreferences.addSearch(city);


        // Prototype scope - create new query
        WeatherQuery query = applicationContext.getBean(WeatherQuery.class);


        // Singleton - get weather data
        WeatherRecord weather = weatherService.getWeather(city);


        Map<String, Object> response = new HashMap<>();
        response.put("weather", weather);
        response.put("requestInfo", requestLogger.getRequestInfo());
        response.put("recentSearches", userPreferences.getRecentSearches());
        response.put("queryInfo", Map.of(
            "queryId", query.getQueryId(),
            "queryTime", query.getQueryTime()
        ));
        response.put("totalRequests", weatherService.getTotalRequests());


        return response;
    }


    @PostMapping("/{city}")
    public Map<String, Object> updateWeather(
            @PathVariable String city,
            @RequestParam double temperature,
            @RequestParam String condition) {
        weatherService.updateWeather(city, temperature, condition);
        return getWeather(city);
    }


    @PostMapping("/preferences/unit")
    public Map<String, String> setUnit(@RequestParam String unit) {
        userPreferences.setTemperatureUnit(unit);
        return Map.of(
            "sessionId", userPreferences.getSessionId(),
            "unit", userPreferences.getTemperatureUnit()
        );
    }
}
```

```java
package com.example.weather.service;
import java.time.LocalDateTime;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope("prototype")
public class WeatherQuery {
    private final String queryId = java.util.UUID.randomUUID().toString();
    private final LocalDateTime queryTime = LocalDateTime.now();

    public String getQueryId() { return queryId; }
    public LocalDateTime getQueryTime() { return queryTime; }
}


package com.example.weather.service;
import java.util.ArrayList;
import java.util.List;
import org.springframework.stereotype.Component;
import org.springframework.web.context.annotation.SessionScope;

@Component
@SessionScope
public class UserWeatherPreferences {
    private final String sessionId = java.util.UUID.randomUUID().toString();
    private final List<String> recentSearches = new ArrayList<>();
    private String temperatureUnit = "Celsius";
    public void addSearch(String city) {
        if (recentSearches.size() >= 5) {
            recentSearches.remove(0);
        }
        recentSearches.add(city);
    }

    public List<String> getRecentSearches() {
        return new ArrayList<>(recentSearches);
    }

    public String getSessionId() { return sessionId; }
    public void setTemperatureUnit(String unit) { this.temperatureUnit = unit; }
    public String getTemperatureUnit() { return temperatureUnit; }
}
```

```java
package com.example.weather.service;
import java.time.LocalDateTime;
import org.springframework.stereotype.Component;
import org.springframework.web.context.annotation.RequestScope;

@Component
@RequestScope
public class WeatherRequestLogger {
    private final String requestId = java.util.UUID.randomUUID().toString();
    private final LocalDateTime requestTime = LocalDateTime.now();
    private String cityRequested;

    public void logRequest(String city) {
        this.cityRequested = city;
    }
    public String getRequestInfo() {
        return String.format("Request ID: %s, Time: %s, City: %s",
            requestId, requestTime, cityRequested);
    }
}

// WeatherService.java
package com.example.weather.service;
import org.springframework.stereotype.Service;
import java.util.HashMap;
import java.util.Map;
import com.example.WeatherRecord;

@Service  // Singleton by default
public class WeatherService {
    private Map<String, WeatherRecord> weatherData = new HashMap<>();
    private int totalRequests = 0;  // Shared counter for all users
    public void updateWeather(String city, double temperature, String condition) {
        weatherData.put(city, new WeatherRecord(city, temperature, condition));
        totalRequests++;
    }

    public WeatherRecord getWeather(String city) {
        totalRequests++;
        return weatherData.get(city);
    }

    public int getTotalRequests() {
        return totalRequests;
    }
}
```

# DAY 27 (23 October)

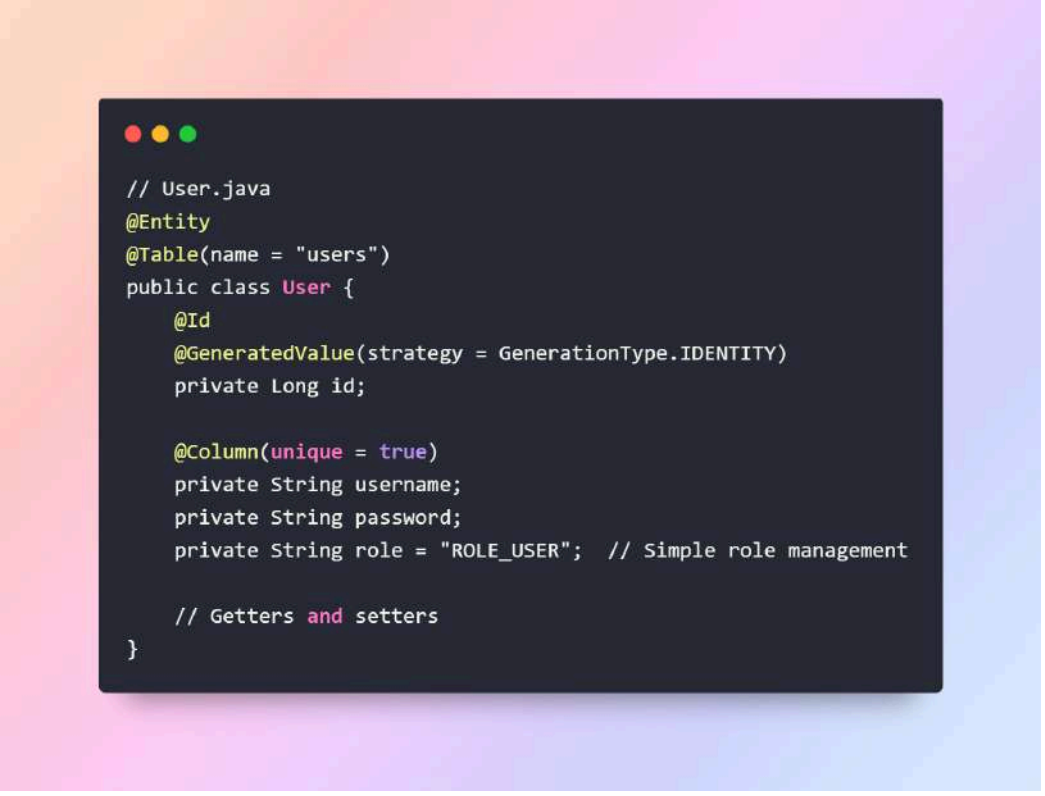
## SPRING SECURITY :




## This flow explains how Spring Boot secures API endpoints using JWT authentication :

1. **Client Request:** The client sends a login request to the authentication controller.
2. **Authentication Controller:** The login request is processed, and credentials are verified using the user repository.
3. User Repository: The repository checks the user credentials for correctness.
4. **Token Generation**: Upon successful verification, a JWT token is generated.
5. **JWT Sent:** The token is returned to the client.
6. **JWT Filter**: Subsequent requests from the client carry the JWT token, which is passed through the JWT filter to check its validity.
7. **Set Security Context**: If the token is valid, the security context is set.
8. **Access Protected Resource**: The client gains access to protected resources after validation.

```java
// JwtUtil.java
@Component
public class JwtUtil {
    @Value("${jwt.secret}")
    private String secret;

    public String generateToken(String username) {
        return Jwts.builder()
            .setSubject(username)
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + 864000000)) // 10 days
            .signWith(SignatureAlgorithm.HS512, secret)
            .compact();
    }

    public String getUsernameFromToken(String token) {
        return Jwts.parser()
            .setSigningKey(secret)
            .parseClaimsJws(token)
            .getBody()
            .getSubject();
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
            return true;
        } catch (Exception e) {
            return false;
        }
    }
}
```



```java
// User.java
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private String username;
    private String password;
    private String role = "ROLE_USER";  // Simple role management

    // Getters and setters
}
```

```java
package com.example;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.filter.OncePerRequestFilter;
import org.springframework.beans.factory.annotation.Autowired;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import java.util.Arrays;
import org.springframework.security.core.context.SecurityContextHolder;

/*
 * Purpose: Intercepts all requests and validates the JWT token for security
 * Responsibilities:
 * 1. Extracts JWT token from the request
 * 2. Validates the token
 * 3. Sets the authentication in the context, so that downstream filters can use it
 * Set up security context if token is valid
 * 4. If validation fails, sends 401 Unauthorized response
 * 5. Otherwise, passes the request to the next filter in the chain
 */
public class JwtFilter extends OncePerRequestFilter {

    @Autowired
    private Jwtutil jwtUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        String token = extractToken(request);
        if (token != null && jwtUtil.validateToken(token)) {
            String username = jwtUtil.getUsernameFromToken(token);
            UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(username, null, Arrays.asList(new SimpleGrantedAuthority("ROLE_USER")));

            SecurityContextHolder.getContext().setAuthentication(authenticationToken);

        }
        filterChain.doFilter(request, response);
    }

    private String extractToken(HttpServletRequest request) {
        String authorizationHeader = request.getHeader("Authorization");
        if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
            return authorizationHeader.substring(7);
        }
        return null;
    }
}
```

```java
// AuthController.java
package com.example.security.controller;

import com.example.security.dto.LoginRequest;
import com.example.security.dto.LoginResponse;
import com.example.security.service.UserService;
import com.example.security.util.JwtUtil;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/auth")
public class AuthController {
    private final UserService userService;
    private final JwtUtil jwtUtil;

    public AuthController(UserService userService, JwtUtil jwtUtil) {
        this.userService = userService;
        this.jwtUtil = jwtUtil;
    }

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest request) {
        try {
            // Validate user
            if (userService.validateUser(request.getUsername(), request.getPassword())) {
                // Get user details
                User user = userService.findByUsername(request.getUsername());

                // Generate token
                String token = jwtUtil.generateToken(user.getUsername());

                // Create response
                LoginResponse response = new LoginResponse(
                    token,
                    user.getUsername(),
                    user.getRole()
                );

                return ResponseEntity.ok(response);
            }

            return ResponseEntity.badRequest().body("Invalid credentials");
        } catch (Exception e) {
            return ResponseEntity.badRequest().body("Login failed: " + e.getMessage());
        }
    }

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody LoginRequest request) {
        try {
            // Create user
            User user = userService.createUser(
                request.getUsername(),
                request.getPassword()
            );

            return ResponseEntity.ok("User registered successfully");
        } catch (Exception e) {
            return ResponseEntity.badRequest()
                .body("Registration failed: " + e.getMessage());
        }
    }
}
```

```java
// UserService.java
package com.example.security.service;

import com.example.security.model.User;
import com.example.security.repository.UserRepository;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional
public class UserService {
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public UserService(UserRepository userRepository, PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }

    public User createUser(String username, String password) {
        // Check if username exists
        if (userRepository.existsByUsername(username)) {
            throw new RuntimeException("Username already exists");
        }

        // Create new user
        User user = new User();
        user.setUsername(username);
        user.setPassword(passwordEncoder.encode(password));

        return userRepository.save(user);
    }

    public User findByUsername(String username) {
        return userRepository.findByUsername(username)
            .orElseThrow(() -> new RuntimeException("User not found"));
    }

    public boolean validateUser(String username, String password) {
        User user = findByUsername(username);
        return passwordEncoder.matches(password, user.getPassword());
    }
}
```

```java
// UserRepository.java
package com.example.security.repository;

import com.example.security.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    // Find user by username
    Optional<User> findByUsername(String username);

    // Check if username exists
    boolean existsByUsername(String username);

    // Find active users by username
    Optional<User> findByUsernameAndActiveTrue(String username);

    // Custom query example
    @Query("SELECT u FROM User u WHERE u.username = :username AND u.active = true")
    Optional<User> findActiveUser(@Param("username") String username);
}
```

```java
// LoginResponse.java
package com.example.security.dto;

public class LoginResponse {
    private String token;
    private String username;
    private String role;

    public LoginResponse(String token, String username, String role) {
        this.token = token;
        this.username = username;
        this.role = role;
    }

    // Getters and setters
    public String getToken() {
        return token;
    }

    public void setToken(String token) {
        this.token = token;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }
}
```

```java
// LoginRequest.java
package com.example.security.dto;

public class LoginRequest {
    private String username;
    private String password;

    // Default constructor
    public LoginRequest() {}

    // Constructor with fields
    public LoginRequest(String username, String password) {
        this.username = username;
        this.password = password;
    }

    // Getters and setters
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

# DAY 28 (24 October)

## Setup :

```
  1 Get-ExecutionPolicy...
  2 Set-ExecutionPolicy AllSigned...
  3 Get-ExecutionPolicy
  4 Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol = [System....
  5 choco --version...
  6 node -v
  7 choco install nodejs-lts --version="20.18.0"
  8 node -v
  9 npm -v
 10 npm install -g typescript
 11 tsc -v
 12 tsc -v
 13 npm install -g @angular/cli
 14 npm fund
 15 ng --version
 16 cd D:\Angular\
 17 ng new myfirstproject...
 18 ng serve -o
 19 cd D:\Angular\myfirstproject
 20 ng serve -o
```

## 1. Angular :

- Angular is a TypeScript-based framework used for building single-page web applications (SPA).
- It is developed by Google and follows a component-based architecture.

## 2. Core Concepts of Angular:

- Modules: Angular apps are modular, and each app is divided into NgModules.
  - Each module represents a block of functionality.
  - The root module is the AppModule.
- Components: The building blocks of Angular apps.
  - Each component consists of:
    - Template (HTML): Defines the view of the component.
    - Class (TypeScript): Contains the logic of the component.
    - Styles (CSS): Defines the component's appearance.
- Services: Classes that handle business logic. Typically used for data sharing and communication between components.
- Directives: Instructions in the DOM. There are two types:
  - Structural Directives: Modify the structure of the DOM (e.g., `*ngIf`, `*ngFor`).
  - Attribute Directives: Modify the appearance or behavior of an element (e.g., `ngClass`, `ngStyle`).
- Data Binding:
  - Interpolation: `{{ }}` syntax, used to display values in templates.
  - Property Binding: `[property]` syntax, used to bind values from the component class to the template.
  - Event Binding: `(event)` syntax, used to handle events (e.g., `(click)`).
  - Two-Way Binding: `[(ngModel)]` syntax, synchronizes data between class and template.
- Routing: Allows navigation between views or pages. The RouterModule is used to define routes.
- Forms: Angular provides template-driven and reactive forms for handling user input.
- Dependency Injection (DI): Angular uses DI to provide instances of classes (like services) where needed.

## 3. Angular Application Structure:

- src/app: Contains the application's root module and component.
- main.ts: The entry point of the application. This file bootstraps the application module.
- index.html: The main HTML file.
- angular.json: Configuration file that controls how the Angular CLI builds your app.

## 4. Angular CLI (Command Line Interface):

- The Angular CLI helps with scaffolding the application and its components, modules, and services.
- Key Commands:
  - `ng new project-name`: Create a new Angular project.
  - `ng serve`: Run the application in development mode.
  - `ng generate component component-name`: Generate a new component.
  - `ng build`: Build the application for production.

## 5. Component Interaction:

- Parent-Child Communication:
  - Use `@Input()` in the child component to receive data from the parent.
  - Use `@Output()` and `EventEmitter` in the child component to send data to the parent.

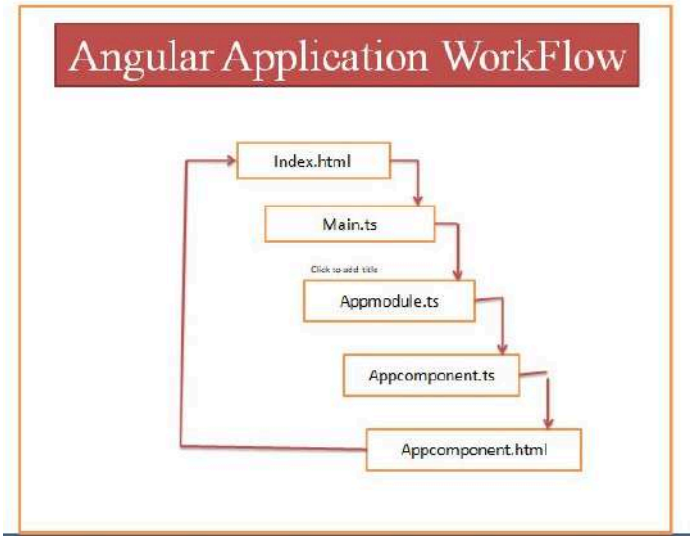
## 6. Lifecycle Hooks:

Angular provides lifecycle hooks that allow you to control your components during its creation and destruction:

- `ngOnInit()`: Called once the component is initialized.
- `ngOnChanges()`: Called when data-bound properties change.
- `ngOnDestroy()`: Called when the component is destroyed.

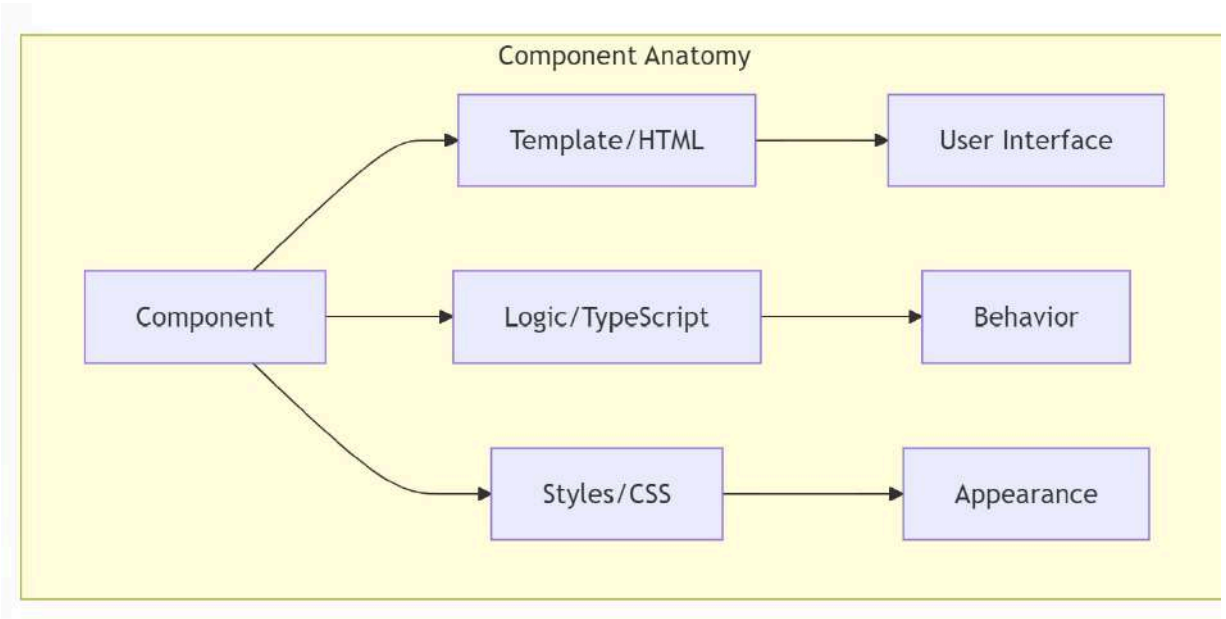
## 7. Angular Flow:

1. App Starts: The app begins with main.ts, which bootstraps the AppModule.
2. AppModule: The root module loads, where the necessary components, services, and other modules are declared.
3. Routing: The app checks for the route in the URL and navigates to the appropriate component.
4. Component Loads: The selected component's template and class are loaded and bound.
5. Rendering: The browser renders the final HTML for the user.
6. User Interacts: User interactions (e.g., button clicks) trigger events and Angular handles them via event binding.



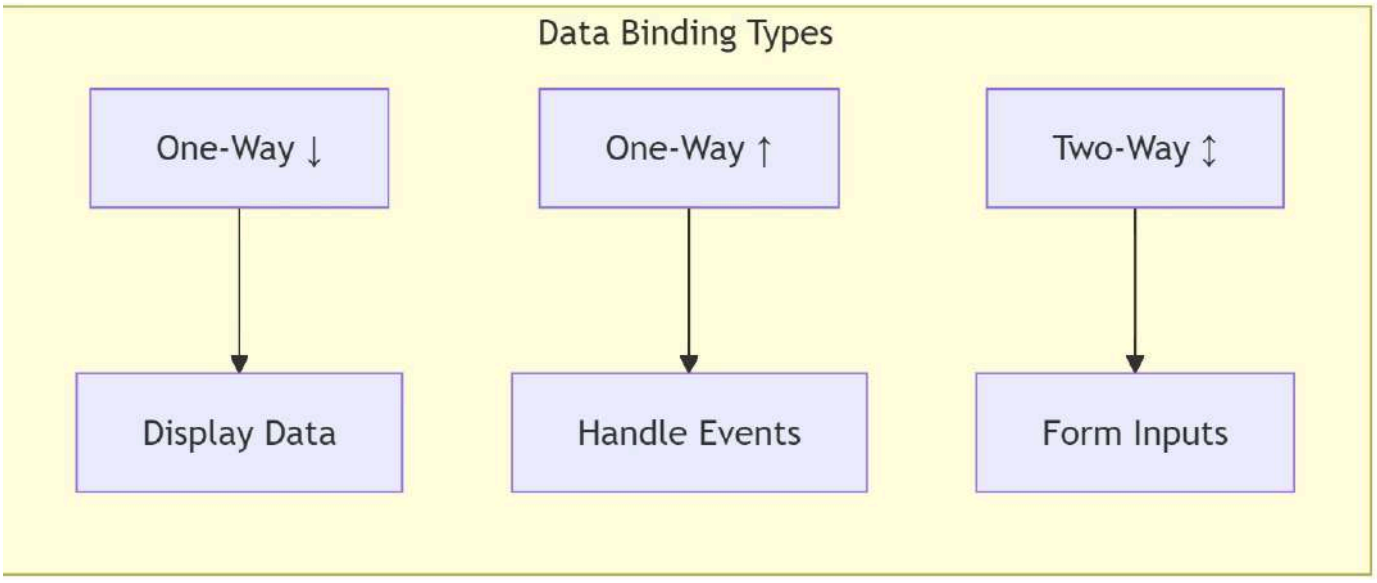

## 8. Advantages of Angular:

- Two-Way Data Binding: Simplifies the interaction between the model and the view.
- Modularity: The application can be divided into smaller, reusable components.
- Routing: Built-in routing to navigate between different views.
- Cross-Platform Development: Can be used for web, mobile (with Ionic), and desktop applications.
- Powerful CLI: The Angular CLI simplifies tasks like setting up projects, testing, and building apps.




## Detailed Explanation:

Think of a Component like a TV set:
The screen (Template) is what users see
The circuit board (Logic) controls how it works
The outer case (Styles) determines how it looks



## Example 1 :

```
// 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!");
  }
}
```

## Example 2 :

```
// 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);
  }
}
```

## Example 3 :

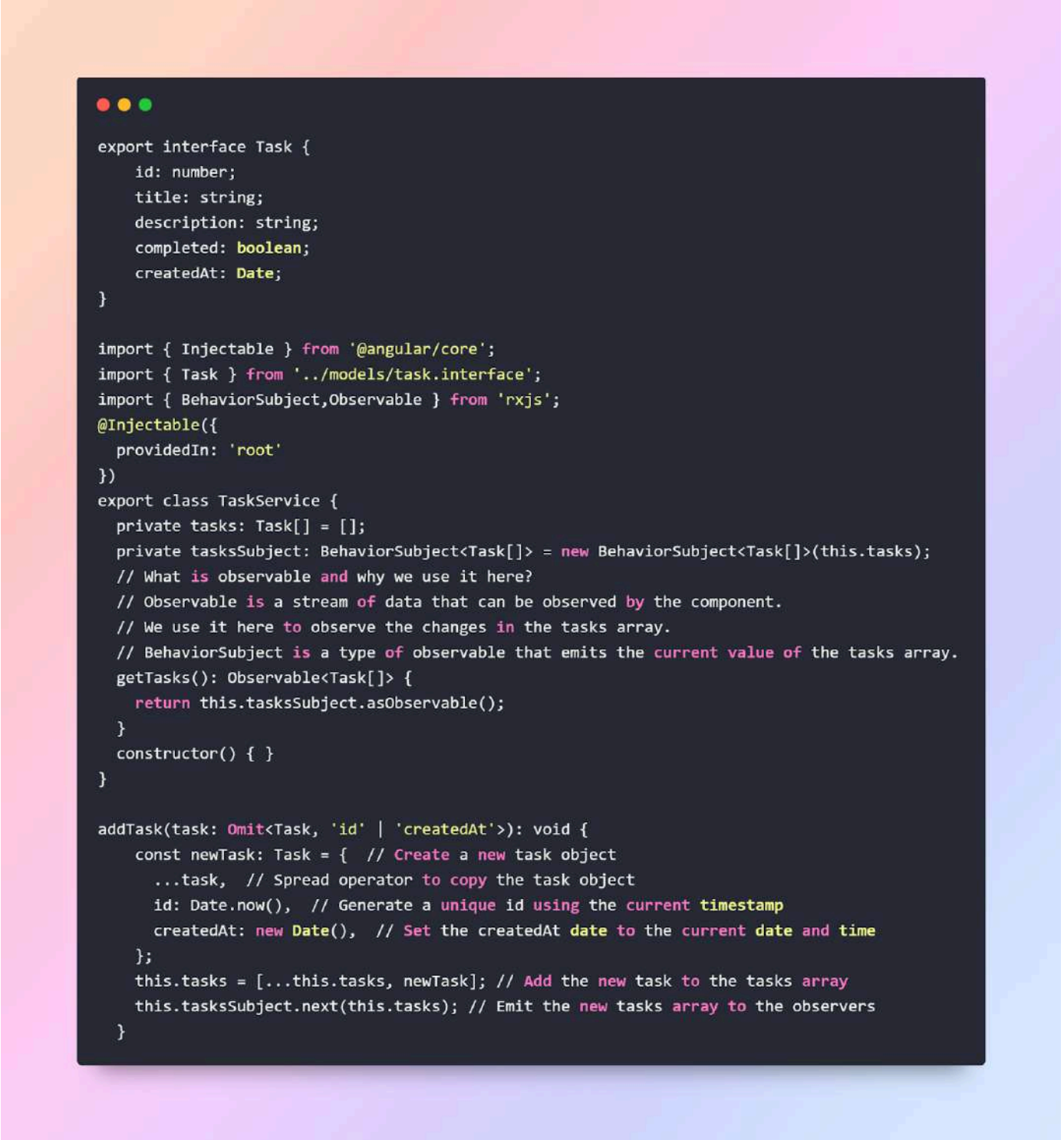
```
// Like instructions: "If box is heavy, get help"
@Component({
  template: `
    <div *ngIf="isHeavy">
      Please get assistance!
    </div>

    <!-- Like "Repeat for each screw" -->
```

```
    <div *ngFor="let item of parts">
      Install {{item}}
    </div>
  `
})
```

## Example 4 :

```
// Like "Paint this part red if it's important"
@Component({
  template: `
    <div [ngStyle]="{'color': isImportant ? 'red' : 'black'}">
      Important Note
    </div>
  `
})
```

# DAY 29 (25 October)



```
export interface Task {
    id: number;
    title: string;
    description: string;
    completed: boolean;
    createdAt: Date;
}

import { Injectable } from '@angular/core';
import { Task } from '../models/task.interface';
import { BehaviorSubject,Observable } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class TaskService {
  private tasks: Task[] = [];
  private tasksSubject: BehaviorSubject<Task[]> = new BehaviorSubject<Task[]>(this.tasks);
  // What is observable and why we use it here?
  // Observable is a stream of data that can be observed by the component.
  // We use it here to observe the changes in the tasks array.
  // BehaviorSubject is a type of observable that emits the current value of the tasks array.
  getTasks(): Observable<Task[]> {
    return this.tasksSubject.asObservable();
  }
  constructor() { }
}

addTask(task: Omit<Task, 'id' | 'createdAt'>): void {
    const newTask: Task = {  // Create a new task object
      ...task,  // Spread operator to copy the task object
      id: Date.now(),  // Generate a unique id using the current timestamp
      createdAt: new Date(),  // Set the createdAt date to the current date and time
    };
    this.tasks = [...this.tasks, newTask]; // Add the new task to the tasks array
    this.tasksSubject.next(this.tasks); // Emit the new tasks array to the observers
  }
```

```
import { Component } from '@angular/core';
import { TaskService } from '../../services/task.service';
import { FormsModule } from '@angular/forms';
import { NgModel } from '@angular/forms';
import { CommonModule } from '@angular/common';
@Component({
  selector: 'app-task-form',
  standalone: true,
  imports: [FormsModule, CommonModule],
  template: `<div class="task-form">
    <h2>Add Task</h2>
    <form (ngSubmit)="onSubmit()"  #taskForm="ngForm">
      <div class="form-group">
      <input type="text" [(ngModel)]="title" name="title" placeholder="Task Title" required class="form-control">

      </div>
      <div class="form-group">
        <textarea [(ngModel)]="description" name="description" placeholder="Task Description" required class="form-control">
        </textarea>
      </div>

      <button type="submit" [disabled]="!taskForm.invalid">Add Task</button>

    </form>
  </div>`,
  styles: [`
    .task-form {
      max-width: 500px;
      margin: 20px auto;
      padding: 20px;
    }
    .form-group {
      margin-bottom: 15px;
    }
    .form-control {
      width: 100%;
      padding: 10px;
      font-size: 16px;
    }
    button {
      background-color: #007bff;
      color: white;
      padding: 10px 20px;
      border: none;
      cursor: pointer;
    }
    button:disabled {
      background-color: #ccc;
      cursor: not-allowed;
    }
  `]
})
//Detail comment about html code written in the template section
//The template section contains a form with two input fields for the task title and description, and a submit button.
//The form is bound to the component's title and description properties using Angular's two-way data binding syntax.
//The form also uses Angular's reactive forms syntax to create a form object that can be used to validate the form data.
//The form is given a name of "taskForm" using the #taskForm syntax, which allows us to access the form object in the component's code.
//The submit button is disabled if the form is invalid, which is determined by the !taskForm.invalid expression.

export class TaskFormComponent {
   title: string = '';
   description: string = '';

   constructor(private taskService: TaskService) {}
   onSubmit():void {
    if (this.title.trim()){
      this.taskService
      .addTask({title: this.title, description: this.description,completed: false});
      this.title = '';
      this.description = '';
      //Detail comment about the class taskform component
      //The taskform component is a simple form that allows the user to add a new task to the task list.
      //The form has two input fields for the task title and description, and a submit button.
      //The form is bound to the component's title and description properties using Angular's two-way data binding syntax.
      //The form also uses Angular's reactive forms syntax to create a form object that can be used to validate the form data.
      //The form is given a name of "taskForm" using the #taskForm syntax, which allows us to access the form object in the component's code.
      //The submit button is disabled if the form is invalid, which is determined by the !taskForm.invalid expression.
    }
  }
}
```

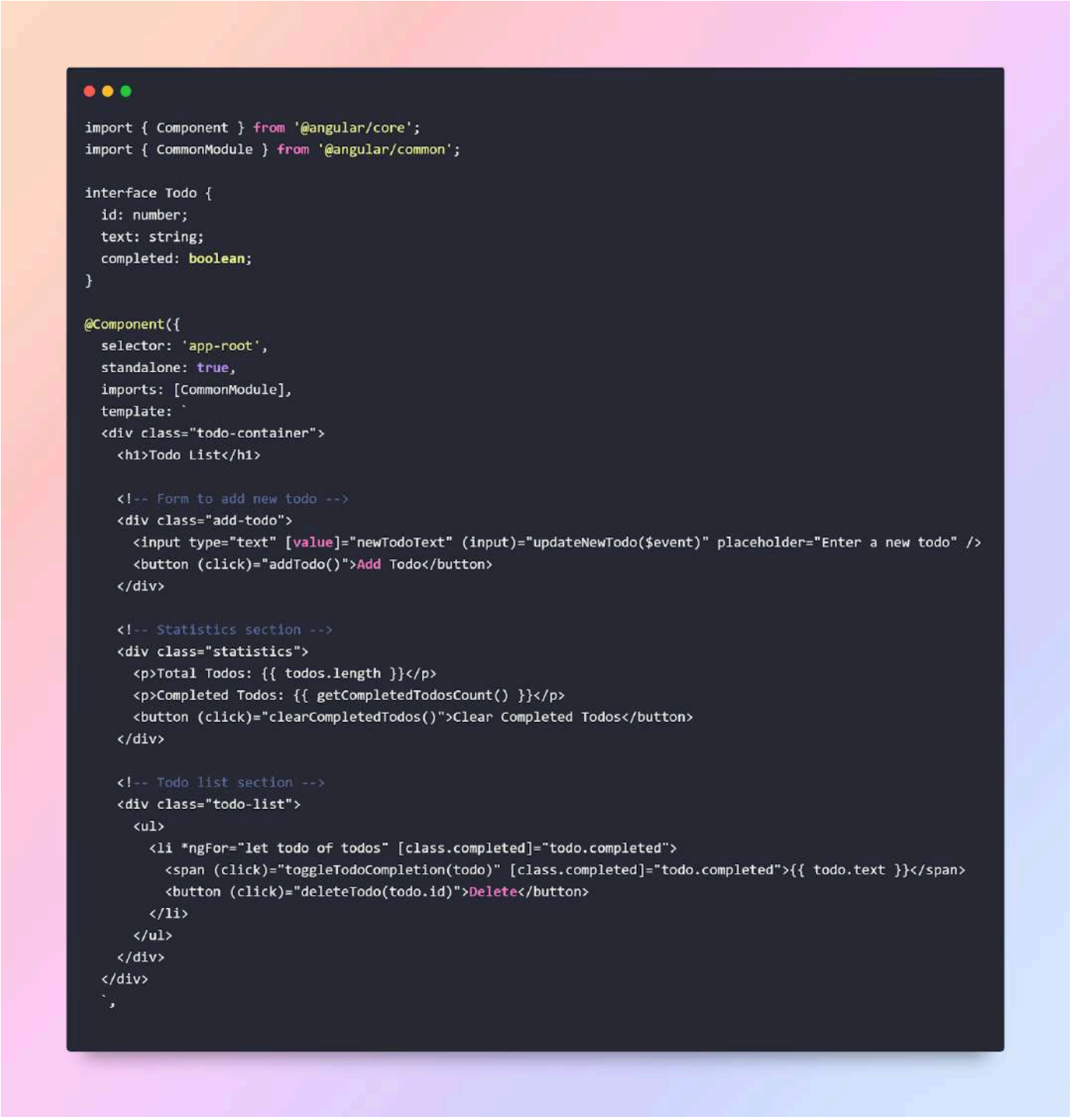
```
import { Component, Input } from '@angular/core';
import { Task } from '../../models/task.interface';
import { TaskService } from '../../services/task.service';
import { FormsModule } from '@angular/forms';
import { DatePipe } from '@angular/common';
@Component({
  selector: 'app-task-item',
  standalone: true,
  imports: [FormsModule,DatePipe],
  template: `<div class="task-item" [class.completed]="task.completed">
    <div class="task-content">
      <input type="checkbox" [(ngModel)]="task.completed" (change)="onToggle(task)">
      <div class="task-text">
        <h3>{{ task.title }}</h3>
        <p>{{ task.description }}</p>
        <small>{{ task.createdAt | date:'short' }}</small>
      </div>
    </div>
    <button (click)="onDelete(task)" class="delete-btn">Delete</button>
  </div>`,
  styles: [`
    .task-item {
      display: flex;
      justify-content: space-between;
      align-items: center;
      padding: 10px;
      border-bottom: 1px solid #ccc;
      background-color: #f0f0f0;
    }
    .task-content {
      display: flex;
      align-items: center;
      gap: 10px;
    }
    .completed {
      background-color: #e0e0e0;
      .task-text {
        opacity: 0.5;
        text-decoration: line-through;
        color: #888;
      }
    }
    .delete-btn {
      background-color: #ff4500;
      color: #fff;
      border: none;
      padding: 5px 10px;
      cursor: pointer;
    }
  `]
})
export class TaskItemComponent {
  @Input() task!: Task;

  constructor(private taskService: TaskService) {}

  onToggle(task: Task) {
    this.taskService.toggleTask(this.task.id);
  }

  onDelete(task: Task) {
    this.taskService.deleteTask(this.task.id);
  }
}
```

```
// Importing required Angular core modules and interfaces
import { Component, OnInit } from '@angular/core';
// Observable for handling asynchronous data streams
import { Observable } from 'rxjs';
// Task interface that defines the structure of a task object
import { Task } from '../../models/task.interface';
// Service that handles task-related operations
import { TaskService } from '../../services/task.service';

@Component({
  // Component's selector used in other templates as <app-task-list>
  selector: 'app-task-list',

  // Template definition using template literal syntax
  template: `
    <!-- Main container for the task list -->
    <div class="task-list">
      <!-- Form component for adding new tasks -->
      <app-task-form></app-task-form>

      <!-- Container for displaying tasks -->
      <div class="tasks">
        <h3>My Tasks</h3>

        <!-- Using ng-container with async pipe to handle Observable -->
        <!-- 'tasks$ | async as tasks' subscribes to the Observable and assigns the value to 'tasks' -->
        <ng-container *ngIf="tasks$ | async as tasks">
          <!-- Iterate over tasks array using ngFor -->
          <!-- Create a task-item component for each task -->
          <app-task-item
            *ngFor="let task of tasks"
            [task]="task"  <!-- Pass task data to child component -->
          ></app-task-item>

          <!-- Show message when no tasks exist -->
          <p *ngIf="tasks.length === 0" class="no-tasks">
            No tasks yet! Add some tasks above.
          </p>
        </ng-container>
      </div>
    </div>
  `,

  // Component-specific styles
  styles: [`
    /* Center the task list and set maximum width */
    .task-list {
      max-width: 800px;
      margin: 0 auto;
      padding: 20px;
    }

    /* Add spacing between form and task list */
    .tasks {
      margin-top: 20px;
    }

    /* Style for empty state message */
    .no-tasks {
      text-align: center;
      color: #6c757d;  /* Grey color for secondary text */
      margin-top: 20px;
    }
  `]
})

// Component class implementation
export class TaskListComponent implements OnInit {
  // Observable that will hold the stream of tasks
  // '!' is the non-null assertion operator indicating it will be initialized
  tasks$!: Observable<Task[]>;

  // Inject TaskService through constructor
  constructor(private taskService: TaskService) {}

  // Lifecycle hook that runs when component initializes
  ngOnInit(): void {
    // Get tasks Observable from service
    this.tasks$ = this.taskService.getTasks();
  }
}
```

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'angularfirst';
  bio = 'I am a software engineer';
  tasks = ['task1', 'task2', 'task3'];
}

<div class=""task-container>
  <h3>Tasks</h3>
  <ul>
    <li *ngFor="let task of tasks">{{tasks}}</li>
  </ul>
</div>

.task-container {
  background-color: #f0f0f0;
  padding: 20px;
  border-radius: 10px;
  margin: 20px;
}

.task-container ul {
  list-style-type: none;
  padding: 0;
}

.task-container li {
  margin-bottom: 10px;
  border-bottom: 1px solid #ccc;
}

import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ChildComponent } from './child.component';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule,ChildComponent],
  templateUrl: './app.component.html'
})
export class AppComponent {
  parentMessage = 'Jinesh is trying to teach Angular';

  handleResponse(response:string){
    this.parentMessage = response;
  }
}

import { Component,Input,Output,EventEmitter } from '@angular/core';

@Component({
  selector: 'app-child',
  standalone: true,
  template: <div><h2>{{message}}</h2> <button (click)="sendResponse()">Send Response to parent</button></div>
})
export class ChildComponent {
  @Input() message: string = '';
  @Output() response = new EventEmitter<string>();

  sendResponse(){
    this.response.emit('Response from child class');
  }
}
```

# DAY 30 (28 October)

## To-do Application:



```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';

interface Todo {
  id: number;
  text: string;
  completed: boolean;
}

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule],
  template: `
  <div class="todo-container">
    <h1>Todo List</h1>

    <!-- Form to add new todo -->
    <div class="add-todo">
      <input type="text" [value]="newTodoText" (input)="updateNewTodo($event)" placeholder="Enter a new todo" />
      <button (click)="addTodo()">Add Todo</button>
    </div>

    <!-- Statistics section -->
    <div class="statistics">
      <p>Total Todos: {{ todos.length }}</p>
      <p>Completed Todos: {{ getCompletedTodosCount() }}</p>
      <button (click)="clearCompletedTodos()">Clear Completed Todos</button>
    </div>

    <!-- Todo list section -->
    <div class="todo-list">
      <ul>
        <li *ngFor="let todo of todos" [class.completed]="todo.completed">
          <span (click)="toggleTodoCompletion(todo)" [class.completed]="todo.completed">{{ todo.text }}</span>
          <button (click)="deleteTodo(todo.id)">Delete</button>
        </li>
      </ul>
    </div>
  </div>
  `,
```

```
  styles: `
    .todo-container {
      max-width: 600px;
      margin: 0 auto;
      padding: 20px;
      border: 1px solid #ccc;
      border-radius: 5px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
    .add-todo {
      display: flex;
      justify-content: space-between;
      margin-bottom: 20px;
    }
    .todo-list {
      margin-top: 20px;
    }
    .statistics {
      margin-top: 20px;
      font-size: 1.2em;
      color: #333;
    }
    .todo-list li {
      display: flex;
      justify-content: space-between;
      align-items: center;
      margin-bottom: 10px;
      padding: 10px;
      border: 1px solid #ccc;
      border-radius: 5px;
      cursor: pointer;
    }
    .todo-list li.completed {
      background-color: #f0f0f0;
      color: #888;
      text-decoration: line-through;
    }
    .todo-list li span.completed {
      text-decoration: line-through;
      color: #888;
    }
    .todo-list button {
      background-color: #ff4d4d;
      border: none;
      color: white;
      padding: 5px 10px;
      border-radius: 3px;
      cursor: pointer;
    }
    .todo-list button:hover {
      background-color: #ff3333;
    }
  `
})
```

```ts
export class AppComponent {
  todos: Todo[] = [
    { id: 1, text: 'Learn Angular', completed: false },
    { id: 2, text: 'Build a project', completed: false },
    { id: 3, text: 'Deploy the project', completed: false }
  ];

  newTodoText: string = '';
  nextId: number = 4;

  updateNewTodo(event: Event): void {
    const input = event.target as HTMLInputElement;
    this.newTodoText = input.value;
  }

  addTodo(): void {
    if (this.newTodoText.trim() === '') {
      return;
    }
    this.todos.push({ id: this.nextId++, text: this.newTodoText, completed: false });
    this.newTodoText = '';
  }

  deleteTodo(id: number): void {
    this.todos = this.todos.filter(todo => todo.id !== id);
  }

  toggleTodoCompletion(todo: Todo): void {
    todo.completed = !todo.completed;
  }

  clearCompletedTodos(): void {
    this.todos = this.todos.filter(todo => !todo.completed);
  }

  getCompletedTodosCount(): number {
    return this.todos.filter(todo => todo.completed).length;
  }
}
```

# PROJECT WORK (29 OCT - 13 NOV)

## Github link :

https://github.com/ashishasnani27/finalProject.git