writing a Java program that performs basic CRUD (Create, Read, Update, Delete) operations and interacts with a user interface (UI) form for input.

**1. Class Definition and Initialization:**

public class TableManager {

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

**Explanation**

**=> Here, you define the TableManager class, which will manage our records.**

**=>You also initialize a static HashMap called table to store the records, where each record is associated with a unique integer ID (Integer) and a string value (String).**

**2. Main Method:**

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int choice;

**Explanation:**

**=> The main method is the entry point of your program.**

**=> You initialize a Scanner object to read user input from the console.**

**=> A variable choice is declared to capture the user's menu selection.**

**3. Menu Loop:**

do {

System.out.println("\n1. Add Record");

// Other menu options...

choice = scanner.nextInt();

scanner.nextLine();

**Explanation:**

**=> A do-while loop is used to repeatedly display the menu until the user decides to exit.**

**=> The menu offers options to add, update, delete, or retrieve a record, or to exit the program.**

**=> User input is captured using scanner.nextInt() for numerical choices, and scanner.nextLine() to handle the input correctly.**

**4.Switch-Case for Menu Choices:**

switch (choice) {

case 1 -> addRecord(scanner);

// Other cases...

**Explanation:**

**=> A switch statement is used to handle different user choices.**

**=> Depending on the user’s input, a corresponding method (addRecord, => updateRecord, etc.) is called to perform the action.**

**5. Add Record:**

private static void addRecord(Scanner scanner) {

int id = scanner.nextInt();

scanner.nextLine();

String record = scanner.nextLine();

table.put(id, record);

**Explanation:**

**=> The addRecord method prompts the user for an ID and a record, then adds this key-value pair to the table HashMap.**

**=> A confirmation message is displayed to indicate that the record was successfully added.**

**6.Update Record:**

private static void updateRecord(Scanner scanner) {

int id = scanner.nextInt();

scanner.nextLine();

if (table.containsKey(id)) {

String newRecord = scanner.nextLine();

table.put(id, newRecord);

}

**Explanation:**

**=> The updateRecord method first checks if the ID exists in the table.**

**=> If it exists, the record is updated with new data provided by the user.**

**=> A confirmation message is shown, or an error message if the ID doesn’t exist.**

**7. Delete Record:**

private static void deleteRecord(Scanner scanner) {

int id = scanner.nextInt();

scanner.nextLine();

if (table.remove(id) != null) {

System.out.println("Record deleted successfully.");

}

**Explanation:**

**=> The deleteRecord method attempts to remove a record by its ID.**

**=> If the record exists and is deleted, a success message is displayed; otherwise, an => error message is shown.**

**8. Get Record:**

private static void getRecord(Scanner scanner) {

int id = scanner.nextInt();

scanner.nextLine();

System.out.println("Record: " + table.getOrDefault(id, "Record with ID " + id + " not found."));

**Explanation:**

**=> The getRecord method retrieves and displays a record by its ID.**

**=> If the ID does not exist, a "not found" message is shown.**

**9. Loop Termination:**

} while (choice != 5);

scanner.close();

**Explanation:**

**=> The loop continues until the user selects the 'Exit' option (choice 5).**

**=> The scanner.close() statement ensures that the Scanner resource is properly closed.**