Intermediate Software Development – MiniProject 4

Project Description:

The goal of this project is to demonstrate the Java Database Connectivity connected to MYSQL database called itm411db. We need to create and populate the database with scripts contained in the given files. Then we need to create and display a command line menu which can be selected by the user to perform basic CRUD operations such as create an Employee record, retrieve data using Employee record by ID, retrieve all employee records, update an Employee record by ID, delete an Employee record by ID from the database itm411db.

Installation, Compile and Run Time requirements:

1. NetBeans IDE 7.2.1
2. Java 1.7.0_11, Java Hotspot(TM) 64-Bit Server VM 23.6-b04
3. Windows 7 version 6.1 running on amd64
4. MySQL – and the jar "mysql-connector-java-5.1.24-bin.jar"

Insights, Expected results, and Challenges:

Insights:

This project helped me get experience on the concepts of Java database connectivity and to connect to MYSQL database to perform required SQL operations.

To complete the project I installed mysql server and created a database named itm411db into which inserted the required tables out of which I will be using Employee table. After this using netbeans I established the connection to the database and performed basic SQL operations such as to create, delete, update and retrieve the Employee record. Thus learned about the use of database and its connectivity.

The packages used in this project are:

1. domain.
2. utilities.
3. driver.

1) **domain** package consists of the following classes:

I. Employee Record

I. Employee Record:

The fields of this class are used to store information about the employee record. This class does the following:

- ☐ Import the required packages.
- ☐ Declaration of variables to store info about the employee record.
- ☐ Providing No-arguments constructor.
- ☐ Providing Full-arguments constructor.

- Providing accessors and mutators for the variables declared.
- Providing toString() method.

2) **utilities** package consist of the following class:

I. JDBCUtilities.

II. UtilityFunctions.

I. JDBCUtilities :

This class contains all the functions related to database. The functions implemented and its functionalities in this class are as follows:

- getConnection() :

  This function establishes the connection with the database itm411db. This function is called before performing any required CRUD operation with the database.
- CreateRecord():

  This method is called whenever the user wants to insert an employee record into database. This function is invoked when key ‘c’ is pressed from the command line menu. The function accepts the inputs of the user for all database table columns to inert the data. All the data should be of required data type otherwise the appropriate error message is displayed. On success inserts the record into database.
- retrieveRecord():

  This method is called whenever the user wants to retrieve a required employee record. This function is invoked when key ‘r’ is pressed from the command line menu. The user is asked to input the Employee number ID of which he wants the record. If the entered employee number ID is valid then respective employee record data is displayed else appropriate error message is displayed.
- retrieveAllRecords() :

  This method is used to retrieve all the employees’ records from the database. This function is invoked when key ‘R’ is pressed from the command line menu.
- updateRecord():

  This method is called whenever the user wants to update a required employee record. This function is invoked when key ‘u’ is pressed from the command line menu .The user is asked to input the Employee number ID of which he wants to update the record. If the entered employee number ID is valid then all the columns values are displayed asking if user wants to update the respective column if user says yes then appropriate column data input of required data type is accepted. After getting all the inputs for the columns the appropriate employee record is updated. If error occurs at any stage then appropriate message is displayed.
- deleteRecord():

  This method is called whenever the user wants to delete a required employee record. This function is invoked when key ‘d’ is pressed from the command line menu .The user is asked to input the Employee number ID of which he wants to delete the record. If the entered employee number ID is valid then respective employee record data is deleted from the database table employees else appropriate error message is displayed.

II. UtilityFunctions:

This class is used to implement methods which will be called from driver class. This class helps in minimizing main() methods and understanding of the main class in a better and easy way. The functions implemented in this class are as follows:

- displayMenu() :

  This function is invoked from the driver class. It displays the menu option on the command line menu from which the user can select the option to perform required operation.

- getUserSelection():

  This function is used to accept input from the user related to database operations. It accepts the input from the user and invokes respective function implemented in JDBCUtilities class and completed the operation.

- printOnConsoleandToFile():

  This function is used to display data onto console as well as write data into the file at the same time.

3) **driver** package consist of the following class:

I. MP4 :

This is the main class from where the project execution starts and performs the following operations:

- Displays display menu to be selected by the user.
- Reads the input and perform the appropriate databse operation according to the requirement of the user.
- Continues to perform above operations until the application exits.

Expected Results:

Expected Results	***Test Result***
1. Should display a command line menu selectable by the user. **[c -** Create an *Employee* record r - Retrieve an *Employee* record by ID R - Retrieve all *Employee* records u - Update an *Employee* record by ID d - Delete an *Employee* record by ID]	Pass
2. Should Create an *Employee* record (when user selects c from 1)	Pass
3. Should Retrieve an *Employee* record by ID(when user selects r from 1)	Pass
4. Should Retrieve all *Employee* records (when user selects R from 1)	Pass
5. Should Update an *Employee* record by ID(when user selects u from 1)	Pass

6. Should Delete an *Employee* record by ID (when user selects d from 1)	Pass

Note: Portable test script(mp4.bat) that executes the application is kept in the **output folder** at project top level with jar file and lib folder. mp4out.txt file gets created in the same folder when the bat file is executed.

Challenges Faced:

While coding the project, there were no major challenges as such. But did encounter minor hiccups while implementing the update employee record. First I implemented to accept the whole UPDATE SQL statement from the user. But then I thought all users may not know the SQL statements. So then I changed the program to accept data for each column values the user wants to update. Then I used user input to update the database record.

Also, while accepting the data for Employee number ID for creating, retrieving, updating and deleting the record from database I had forgot to check for the valid Employee number ID and Employee reports to entered by the user. Then I implemented it.

Screenshots:

1. **EmployeeRecord Class**

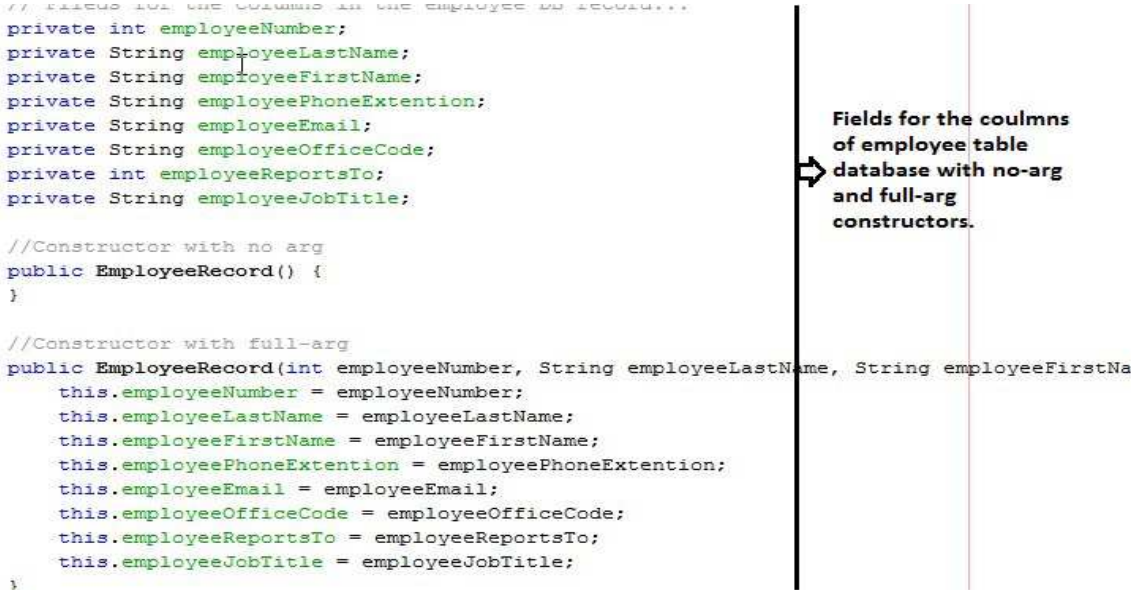
Snapshot 1-1:

```
public String getEmployeePhoneExtention() {
    return employeePhoneExtention;
}

public void setEmployeePhoneExtention(String employeePhoneExtention) {
    this.employeePhoneExtention = employeePhoneExtention;
}

public String getEmployeeEmail() {
    return employeeEmail;
}

public void setEmployeeEmail(String employeeEmail) {
    this.employeeEmail = employeeEmail;
}

public String getEmployeeOfficeCode() {
    return employeeOfficeCode;
}

public void setEmployeeOfficeCode(String employeeOfficeCode) {
    this.employeeOfficeCode = employeeOfficeCode;
}
```

Accessors and mutators for the fields.

Snapshot 1-2:

Snapshot 1-3:



Snapshot 1-4:

```
public int getEmployeeReportsTo() {
    return employeeReportsTo;
}

public void setEmployeeReportsTo(int employeeReportsTo) {
    this.employeeReportsTo = employeeReportsTo;
}

public String getEmployeeJobTitle() {
    return employeeJobTitle;
}

public void setEmployeeJobTitle(String employeeJobTitle) {
    this.employeeJobTitle = employeeJobTitle;
}

//Overridden string method
@Override
public String toString() {
    return "EmployeeRecord{" + "employeeNumber=" + employeeNumber + ", employeeLastName=" + employeeLastName + ", em
}
```

Accessors, Mutators and toString() method of class EmployeeRecord

2. JDBCUtilities Class

Snapshot 2-1:

```
            catch (SQLException ex) {
                Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }// returns boolean result.
    return result;
}

//This function is called to connect to the database itm411db.
private static Connection getConnection() {
    Connection conn = null;
    try {
        //Establishes a connection to the given database URL.
        conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/itm411DB", "root", "admin");
    }//catches exception if error in connection to database occurs.
    catch (SQLException ex) {
        Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
    }
    return conn;
}
}
```

function to establish the connection with the database to perform all the CRUD operations(create, retrieve, delete and update the employee record).

Snapshot 2-2:

```
public static boolean createRecord() {
    //columns present in the database
    String[] fields = {"employeeNumber", "employeeLastName", "employeeFirstName", "employeePhoneExtension", "empl
    String[] types = {"int", "String", "String", "String", "String", "String", "int", "String"};
    // to store the values entered by the user for respective columns.
    String[] fieldValues = new String[fields.length];
    //to determine success or failure of the record addition into database.
    boolean result = false;
    //iterate through the field values to accept all required column values.
    for (int i = 0; i < fields.length; i++) {
        UtilityFunctions.printOnConsoleandToFile("Please enter value for field " + fields[i] + " using data type
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        try {
            //read the data entered by the user.
            String dataEntered = br.readLine();
            UtilityFunctions.printToFile(dataEntered);
            // to check data entered is valid or else perform desired operation.
            if (dataEntered != null) {
                if (dataEntered.trim().equals("")) {
                    fieldValues[i] = "-1";
                } else {
                    fieldValues[i] = dataEntered;
                }
            }
```

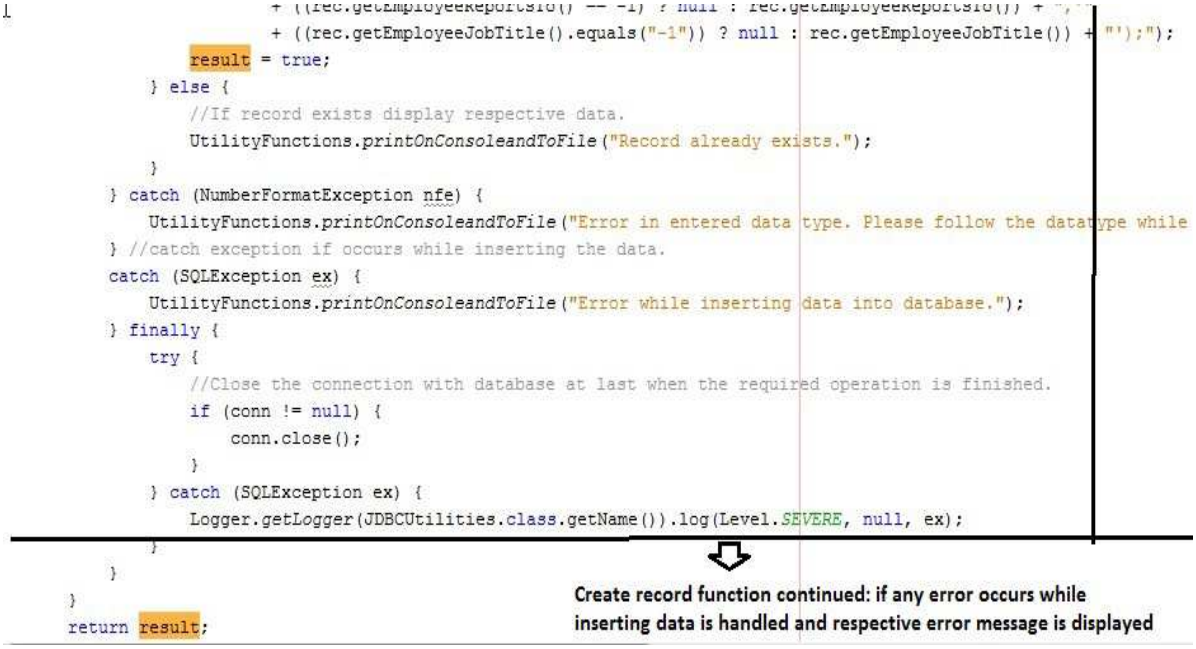
Code to create record in the database. The value entered from the user is read for each columns and validated is shown here.

Snapshot 2-3:

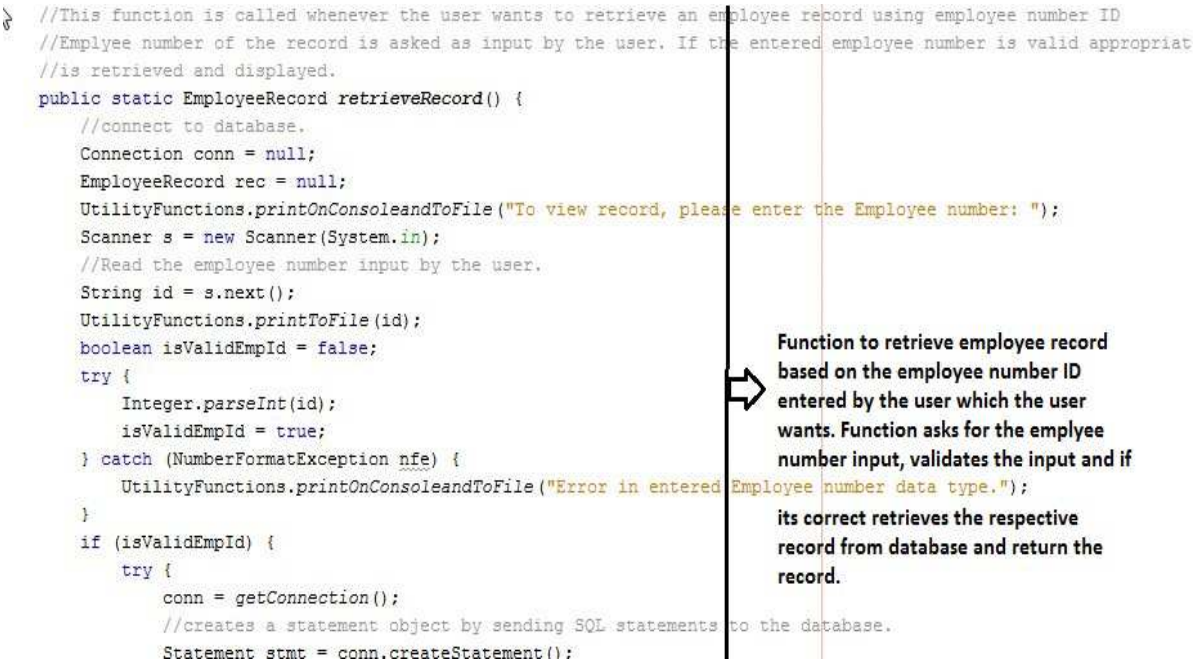
```
    Connection conn = getConnection();
    try {
        //create employee record using the values entered by the user.
        EmployeeRecord rec = new EmployeeRecord(Integer.parseInt(fieldValues[0]), fieldValues[1], fieldValues[2

        //creates a statement object by sending SQL statements to the database.
        Statement stmt = conn.createStatement();
        //Executes the SQL statement. This is to check whether the employee record for the employee number
        //already exists. This check is done to avoid duplicate entry.
        ResultSet rs = stmt.executeQuery("SELECT * FROM Employees WHERE employeeNumber = " + rec.getEmployeeNur
        //if the record not exists then insert the record.
        if (!rs.next()) {
            //Execute theSQL insert statement to insert the data into the database.
            //On success set result true.
            stmt.execute("INSERT INTO employees ("
                    + "employeeNumber, lastName, firstName, extension, "
                    + "email, officeCode, reportsTo, jobTitle)"
                    + " VALUES("
                    + ((rec.getEmployeeNumber() == -1) ? null : rec.getEmployeeNumber()) + ",'"
                    + ((rec.getEmployeeLastName().equals("-1")) ? null : rec.getEmployeeLastName()) + "','"
                    + ((rec.getEmployeeFirstName().equals("-1")) ? null : rec.getEmployeeFirstName()) + "','"
                    + ((rec.getEmployeePhoneExtention().equals("-1")) ? null : rec.getEmployeePhoneExtention())
                    + ((rec.getEmployeeEmail().equals("-1")) ? null : rec.getEmployeeEmail()) + "','"
                    + ((rec.getEmployeeOfficeCode().equals("-1")) ? null : rec.getEmployeeOfficeCode()) + "',"
```

Create record function continued: connection is esatblished with database and the created emplyee record is added into it.

Snapshot 2-4:



Snapshot 2-5:



Snapshot 2-6:

```
            ResultSet rs = stmt.executeQuery("SELECT * FROM Employees WHERE employeeNumber = " + id);
            // Retrieved record values are wrapped into an record object
            while (rs.next()) {
                int employeeNumber = rs.getInt("employeeNumber");
                String employeeLastName = rs.getString("lastName");
                String employeeFirstName = rs.getString("firstName");
                String employeeExtensionNumber = rs.getString("extension");
                String employeeEmailID = rs.getString("email");
                String employeeOfficeCode = rs.getString("officeCode");
                int employeeReportsTo = rs.getInt("reportsTo");
                String employeeJobTitle = rs.getString("jobTitle");
                rec = new EmployeeRecord(employeeNumber, employeeLastName, employeeFirstName, employeeExtensionNur
                        employeeEmailID, employeeOfficeCode, employeeReportsTo, employeeJobTitle);
            }
        }//catch exception if occurs while retrieving the data.
        catch (SQLException ex) {
            //Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            try {
                //Close the connection with database at last when the required operation is finished.
                if (conn != null) {
                    conn.close();
                }
            }//catches exception if error occurs while closing the database connection.
```

Function to retrieve employee record based on the employee number ID continued.

Snapshot 2-7:

```
public static boolean updateRecord() {

    UtilityFunctions.printOnConsoleandToFile("Please enter the Employee number ID of which the Employee record need to b
    Connection conn = null;
    Scanner s = new Scanner(System.in);
    //Read the employee number input by the user.
    String empIDentered = s.next();
    UtilityFunctions.printToFile(empIDentered);
    boolean result = false;
    boolean isValidEmpId = false;

    try {
        //check for valid data type of employee number.
        Integer.parseInt(empIDentered);
        isValidEmpId = true;
    } catch (NumberFormatException nfe) {
        UtilityFunctions.printOnConsoleandToFile("Error in entered Employee number data type.");
    }
    //If valid data typr continue.
    if (isValidEmpId) {
        try {
             //connect to database.
            conn = getConnection();
            //creates a statement object by sending SQL statements to the database.
            Statement stmt = conn.createStatement();
            //Executes the SQL statement. This is to check whether the employee record for the employee number
```

Update record function invoked when key 'u' is pressed. The user is asked for employee number ID after checking for valid ID the function iterates and accepts data for all columns of the database. If all the inputs are correctly entered according to data type respective employee record row is updated in the database.

Snapshot 2-8:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM Employees WHERE employeeNumber = " + empIDentered);
if (rs.next()) {
    //columns present in the database
    String[] fields = {"employeeNumber", "employeeLastName", "employeeFirstName", "employeePhoneExtension", "e
    // columns name present in the database.
    String[] databaseFields = {"employeeNumber", "lastName", "firstName", "extension", "email", "officeCode",
    String[] types = {"int", "String", "String", "String", "String", "String", "int", "String"};
    String[] fieldValues = new String[fields.length];
    fieldValues[0] = empIDentered;
    int noCount, defaultCount;
    noCount = defaultCount = 0;
     //iterate through the field values to accept the required column values to update record.
    for (int i = 1; i < fields.length; i++) {
        UtilityFunctions.printOnConsoleandToFile("Want to update " + fields[i] + " (y/n)? ");
        s = new Scanner(System.in);
        switch (s.next()) {
            case "y":
            case "Y":
                UtilityFunctions.printOnConsoleandToFile("Please enter value for field " + fields[i] + " usin
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                try {
                    //read the data entered by the user.
                    String dataEntered = br.readLine();
                    UtilityFunctions.printToFile(dataEntered);
                    fieldValues[i] = dataEntered.trim();
                }// catch if anu exceptiopn occurs while performing above operations.
                catch (IOException ex) {
```

Update record to database function continued. The code shown here iterates and accepts the value for each columns in the database for employee number ID entered after checking whether the employee number is valid are not.

Snapshot 2-9:

```
            case "n":
            case "N":
                noCount++;
                if (i == 6) {
                    fieldValues[i] = Integer.toString(rs.getInt(databaseFields[i]));
                } else {
                    fieldValues[i] = rs.getString(databaseFields[i]);
                }
                break;
            default:
                defaultCount++;
                if (i == 6) {
                    fieldValues[i] = Integer.toString(rs.getInt(databaseFields[i]));
                } else {
                    fieldValues[i] = rs.getString(databaseFields[i]);
                }
                break;
        }
    }
    //check if values are entered to any field to update if yes continue.
    if (noCount != 7 && defaultCount != 7 && (noCount + defaultCount) != 7) {
        //create employee record using the values entered by the user.
        EmployeeRecord rec = new EmployeeRecord(Integer.parseInt(fieldValues[0]), fieldValues[1], fieldValues[2
        // update the respective employee record with the changed data.
        stmt.execute("UPDATE employees SET "
                + databaseFields[1] + "='" + rec.getEmployeeLastName() + "', "
                + databaseFields[2] + "='" + rec.getEmployeeFirstName() + "', "
                + databaseFields[3] + "='" + rec.getEmployeePhoneExtention() + "', "
                + databaseFields[4] + "='" + rec.getEmployeeEmail() + "', "
```

update record funtion continued. Here, the code checks for the data input of no data value entered for any columns of the database it is checked and handled.

Snapshot 2-10:


```
                + databaseFields[5] + "='" + rec.getEmployeeOfficeCode() + "', "
                + databaseFields[6] + "=" + rec.getEmployeeReportsTo() + ", "
                + databaseFields[7] + "='" + rec.getEmployeeJobTitle() + "'"
                + "WHERE " + databaseFields[0] + "=" + fieldValues[0] + ";");
            result = true;
        } else {
            //if no data entered for any columns display error message.
            UtilityFunctions.printOnConsoleandToFile("No data was entered to any columns to update the employee numb
        }
    } else {
        //If record exists display respective data.
        UtilityFunctions.printOnConsoleandToFile("Record for the employee number " + empIDentered + " does not exist
    }
} catch (NumberFormatException nfe) {
    UtilityFunctions.printOnConsoleandToFile("Error in entered data type. Please follow the datatype while entering
} //catch exception if occurs while inserting the data.
catch (SQLException ex) {
    UtilityFunctions.printOnConsoleandToFile("Error while inserting data into database.");
} finally {
    try {
        //Close the connection with database at last when the required operation is finished.
        if (conn != null) {
            conn.close();
        }
    }//catches exception if error occurs while closing the database connection.
    catch (SQLException ex) {
        Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
    }
```

Snapshot 2-11:


```
            if (!rs.next()) {
                result = true;
            } else {
                result = false;
            }
        }//catches exception while reading data from the result returned from the database query .
        catch (SQLException ex) {
            Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            try {
                //Close the connection with database at last when the required operation is finished.
                if (conn != null) {
                    conn.close();
                }
            }//catches exception if error occurs while closing the database connection.
            catch (SQLException ex) {
                Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    } else {
        try {
            //Close the connection with database at last when the required operation is finished.
            if (conn != null) {
                conn.close();
            }
        }//catches exception if error occurs while closing the database connection.
```

Snapshot 2-12:

```
//This function is called whenever the user wants to delete an employee record.
//Emplyee number of the record is asked as input by the user. If the entered employee number is valid appropriate
//is deleted .
public static boolean deleteRecord() {
    //connect to database.
    Connection conn = null;
    UtilityFunctions.printOnConsoleandToFile("To delete record, please enter the employee number: ");
    Scanner s = new Scanner(System.in);
    //Read the employee number input by the user.
    String id = s.next();
    UtilityFunctions.printToFile(id);
    boolean result = false;
    boolean isValidEmpId = false;
    try {
        Integer.parseInt(id);
        isValidEmpId = true;
    } catch (NumberFormatException nfe) {
        UtilityFunctions.printOnConsoleandToFile("Error in entered Employee number data type.");
    }
    if (isValidEmpId) {
        try {
            conn = getConnection();
            //creates a statement object by sending SQL statements to the database.
            Statement stmt = conn.createStatement();
            //Executes the SQL statement. This is to check whether the employee record for the employee number
            //exists. This check is done to avoid deleting the data which is not present.
            ResultSet rs = stmt.executeQuery("SELECT * FROM Employees WHERE employeeNumber = " + id);
```

Function to delete a employee record based on the employee number ID entered by the user which the user wants. Function asks for the emplyee number input, validates the input and if its correct deletes the respective record from database and returns success or failure message.

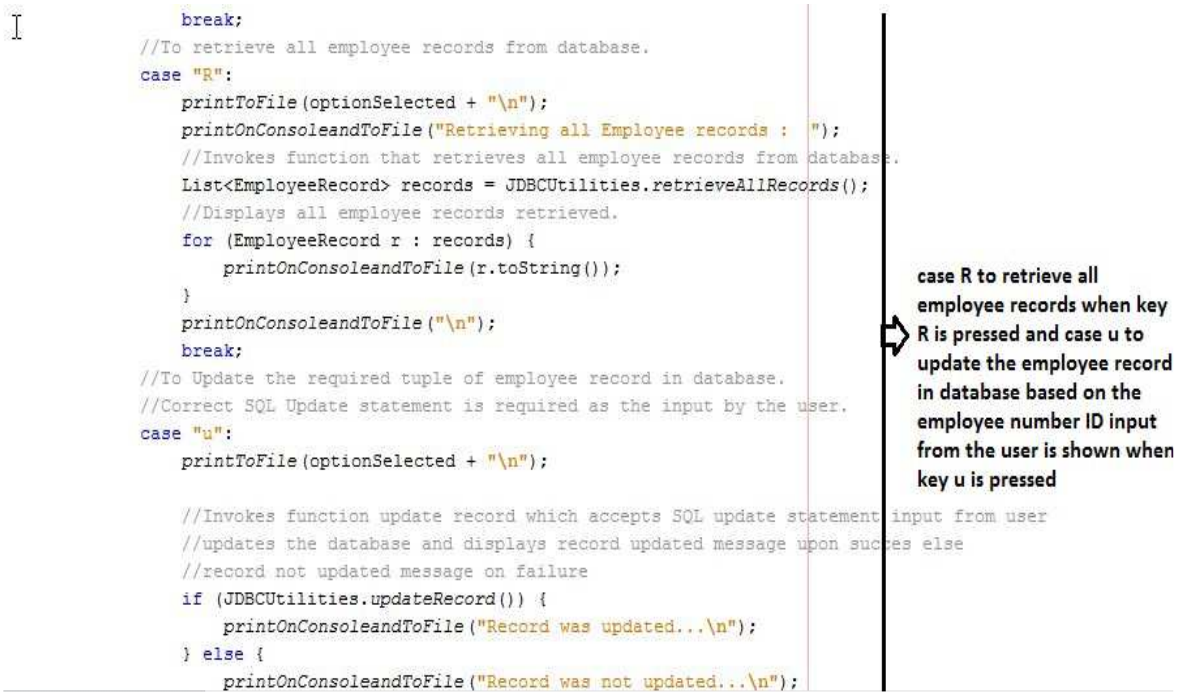
3. UtilityFunctions Class :

Snapshot 3-1:

```
            // Employee number whose employee record to be deleted is accepted as input from the user.
            //Then based on that employee number respective employee record is deleted.
            case "d":
                printToFile(optionSelected + "\n");
                //Invoeks delete function that accepts employee number from the user and
                //deletes the respective employee record from database and displays respective message
                // based on success and failure.
                if (JDBCUtilities.deleteRecord()) {
                    printOnConsoleandToFile("Record was deleted...\n");
                } else {
                    printOnConsoleandToFile("Record was not deleted...\n");
                }

                break;
            //option to exit from the application
            case "q":
                printToFile(optionSelected + "\n");
                printOnConsoleandToFile("Exiting.... :  \n");
                System.exit(0);
                break;
        }
        sc.reset();
    }
}
```

case d which invokes function to delete an employee record from database based on employee id input from the user when key d is pressed is shown here. Also, case q to exit from the application when key q is pressed is shown.

Snapshot 3-2:



Snapshot 3-3:

Snapshot 3-4:

```
public class UtilityFunctions {

    private static FileOutputStream fos = null;
    static boolean initialized = true;

    //displays the options at the console to perform database operations.
    public static void displayMenu() {
        // Selected option will invoke an assitive method outside of this driver class...
        printOnConsoleandToFile("Please enter choice for CRUD operations from the following options: ");
        printOnConsoleandToFile("c - Create an Employee record.\n"
                + "r - Retrieve an Employee record by ID.\n"
                + "R - Retrieve all Employee records.\n"
                + "u - Update an Employee record by ID.\n"
                + "d - Delete an Employee record by ID.\n"
                + "q - Quit the application\n");
    }

    //reads the input out of options selected by the user and perform respective operations
    //Key c option - to Create an Employee record, Key r option - Retrieve an Employee record by ID
    //Key R option - Retrieve all Employee records, Key u option - Update an Employee record by ID
    //Key d option - Delete an Employee record by ID,Key q option - Quit the application
```

Function which displays menu on the command line for user to select the required operation to perform

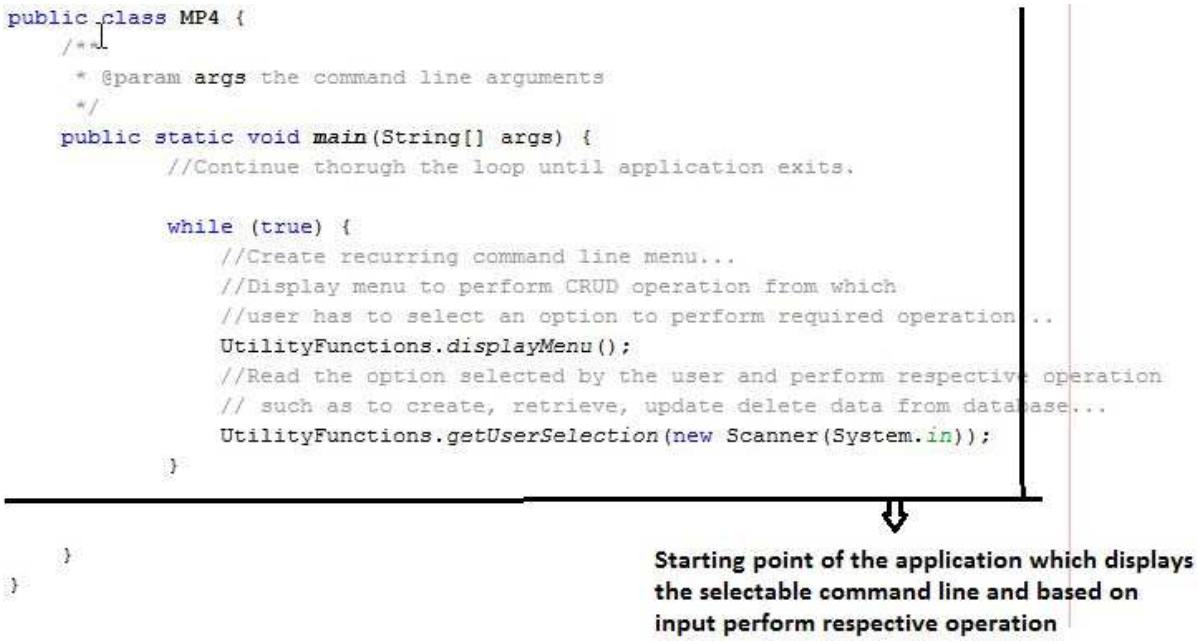
Snapshot 3-5:

```
//Function to display and write respective message onto console and to a file.
public static void printOnConsoleandToFile(String displayData) {
    //Display the data on console
    System.out.println(displayData);
    //execute the statement and catch exception if error occurs.
    try {
        if (initialized) {
            //creating the pointer to mp4out.txt file
            fos = new FileOutputStream("dist/mp4out.txt", false);
            initialized = false;
        }
        //write the data to file name mp4out.txt
        fos.write(displayData.getBytes());
    } catch (IOException ex) {
        Logger.getLogger(UtilityFunctions.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Function to print the required data onto console and mp4out.txt file at the same time.

4. MP4 Class

Snapshot 4-1:

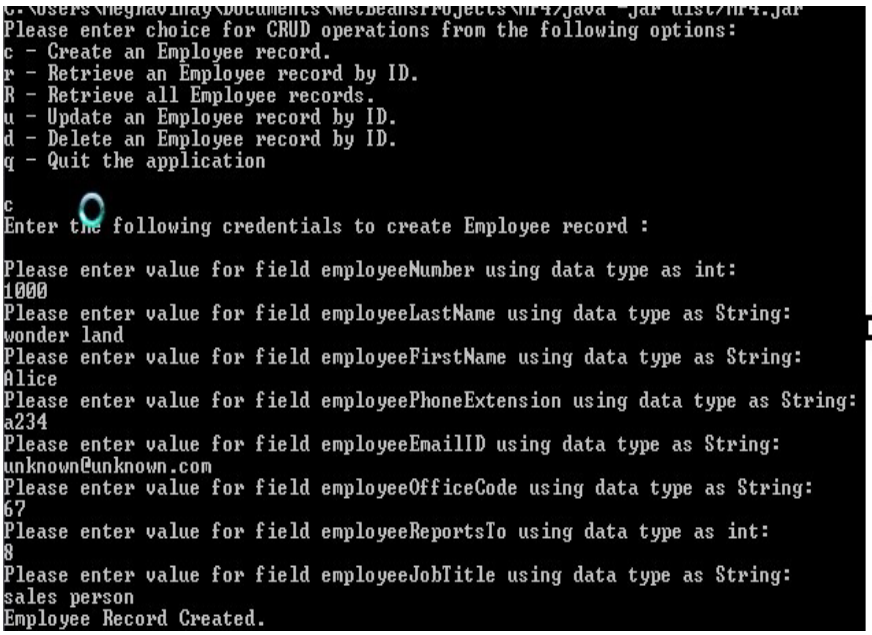


```
public class MP4 {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
            //Continue thorugh the loop until application exits.

            while (true) {
                //Create recurring command line menu...
                //Display menu to perform CRUD operation from which
                //user has to select an option to perform required operation ..
                UtilityFunctions.displayMenu();
                //Read the option selected by the user and perform respective operation
                // such as to create, retrieve, update delete data from database...
                UtilityFunctions.getUserSelection(new Scanner(System.in));
            }

    }
}
```

Starting point of the application which displays the selectable command line and based on input perform respective operation

5. Output

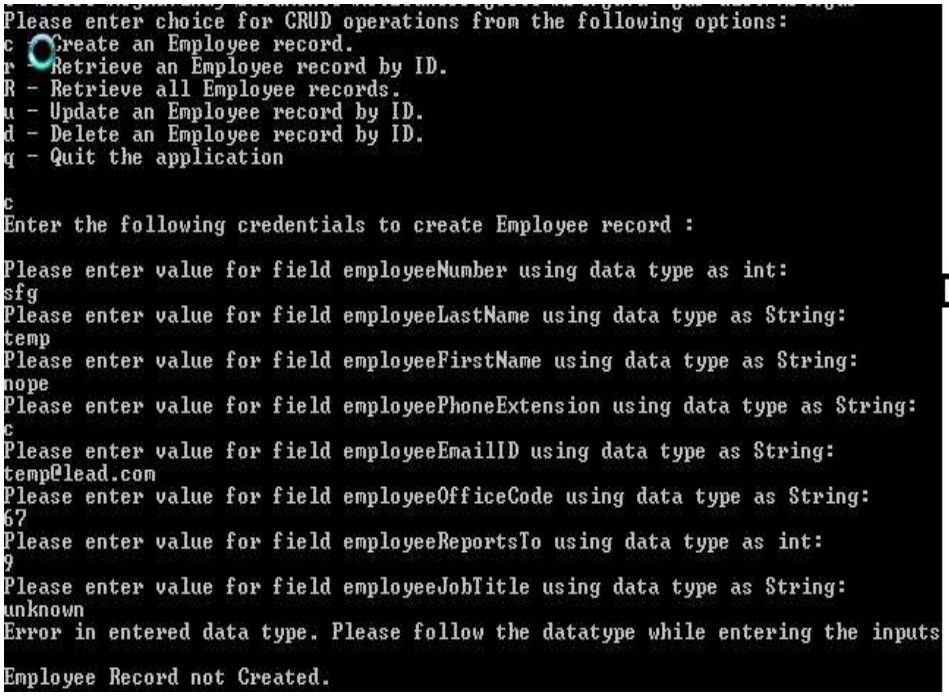
Snapshot 5-1:



```
Please enter choice for CRUD operations from the following options:
c - Create an Employee record.
r - Retrieve an Employee record by ID.
R - Retrieve all Employee records.
u - Update an Employee record by ID.
d - Delete an Employee record by ID.
q - Quit the application

c
Enter the following credentials to create Employee record :

Please enter value for field employeeNumber using data type as int:
1000
Please enter value for field employeeLastName using data type as String:
wonder land
Please enter value for field employeeFirstName using data type as String:
Alice
Please enter value for field employeePhoneExtension using data type as String:
a234
Please enter value for field employeeEmailID using data type as String:
unknown@unknown.com
Please enter value for field employeeOfficeCode using data type as String:
67
Please enter value for field employeeReportsTo using data type as int:
8
Please enter value for field employeeJobTitle using data type as String:
sales person
Employee Record Created.
```

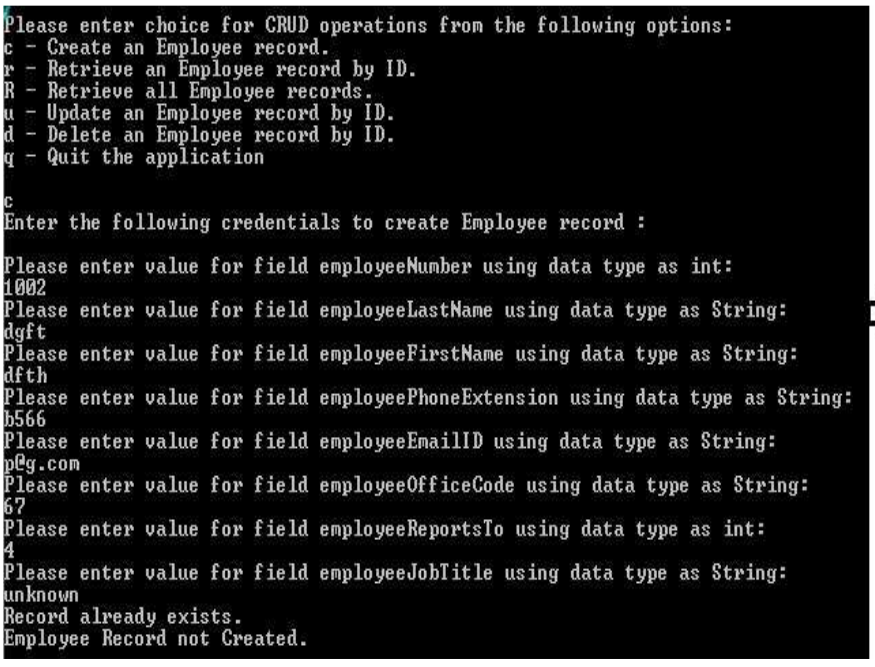
output-> create an employee record in the database when key 'c' is pressed. It asks input for values of each columns of the database and creates the respective record in the database.

Snapshot 5-2:



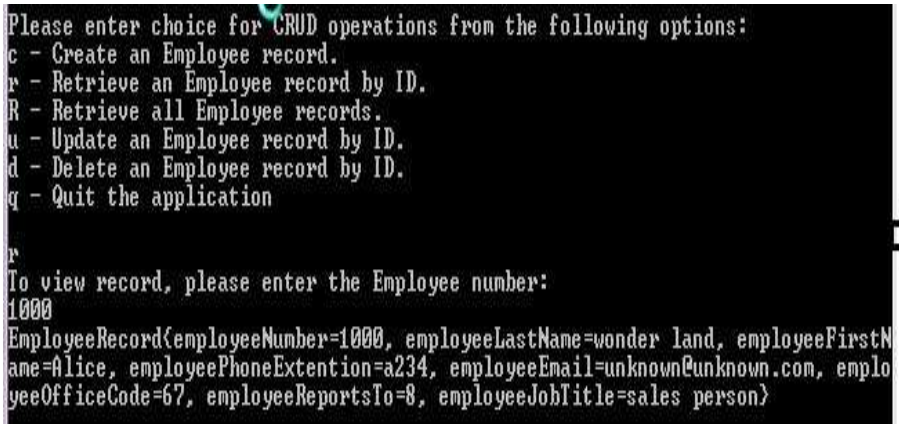
output-> Error message displayed whenever incorrect data type entered other than required for to create a record in the database. Here employee number is given as dtring instead of integer.

Snapshot 5-3:

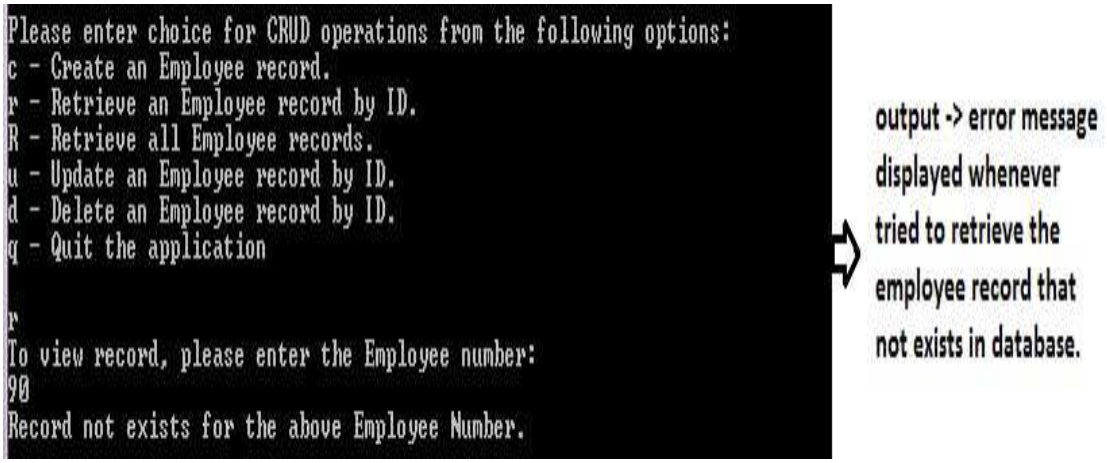


output -> Error message displayed if tried to create the record for an employee number that already exists in the database.

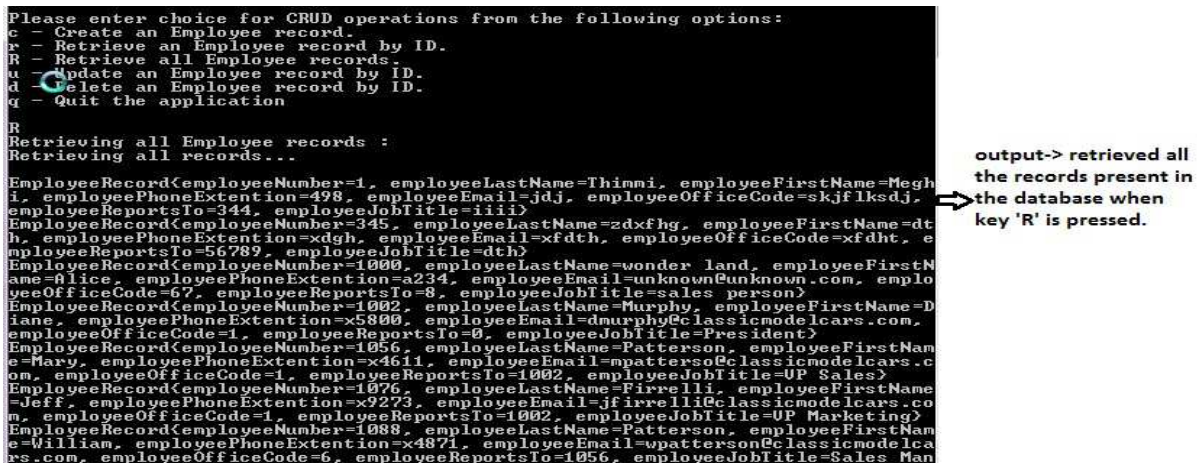
Snapshot 5-4:



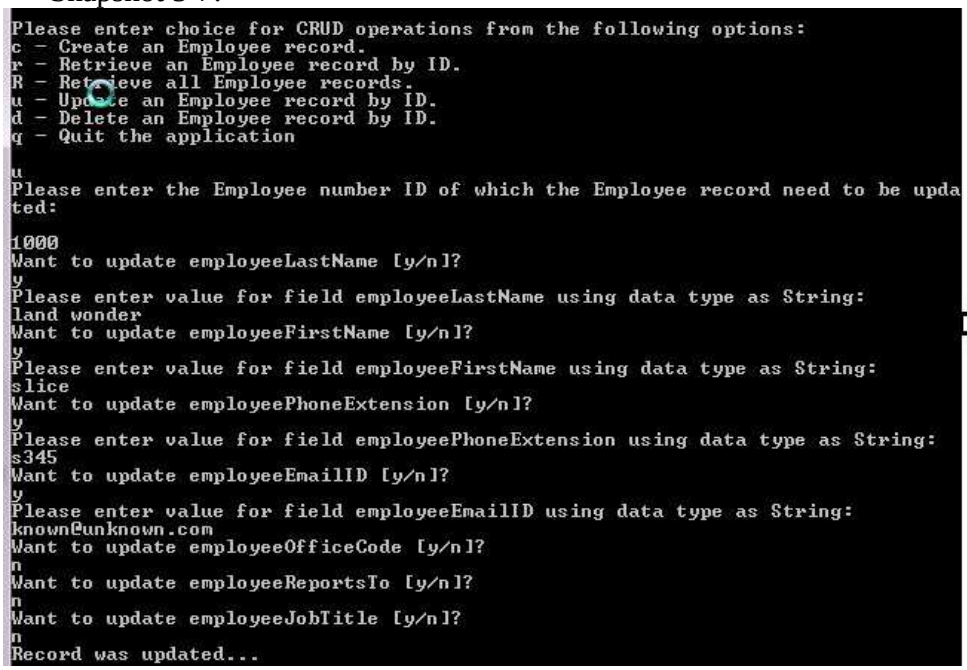
Snapshot 5-5:



Snapshot 5-6:

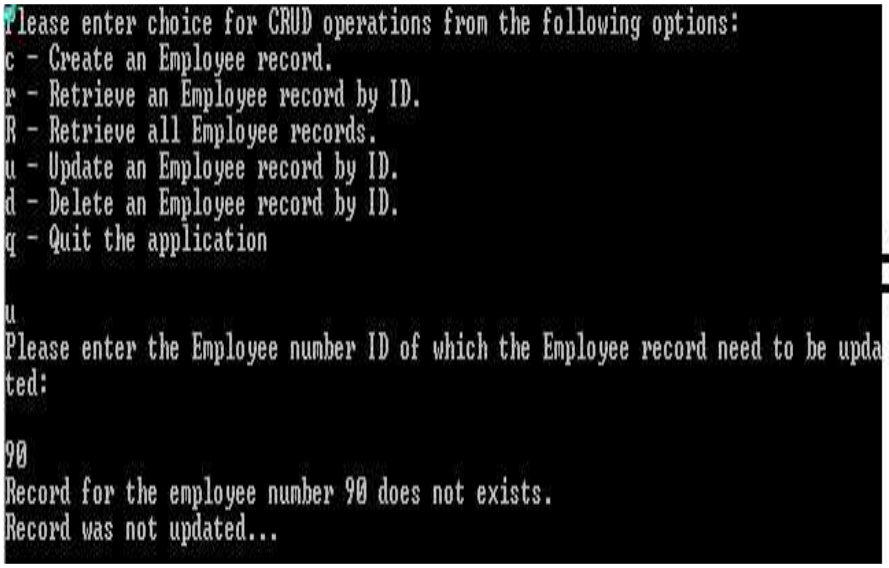


Snapshot 5-7:



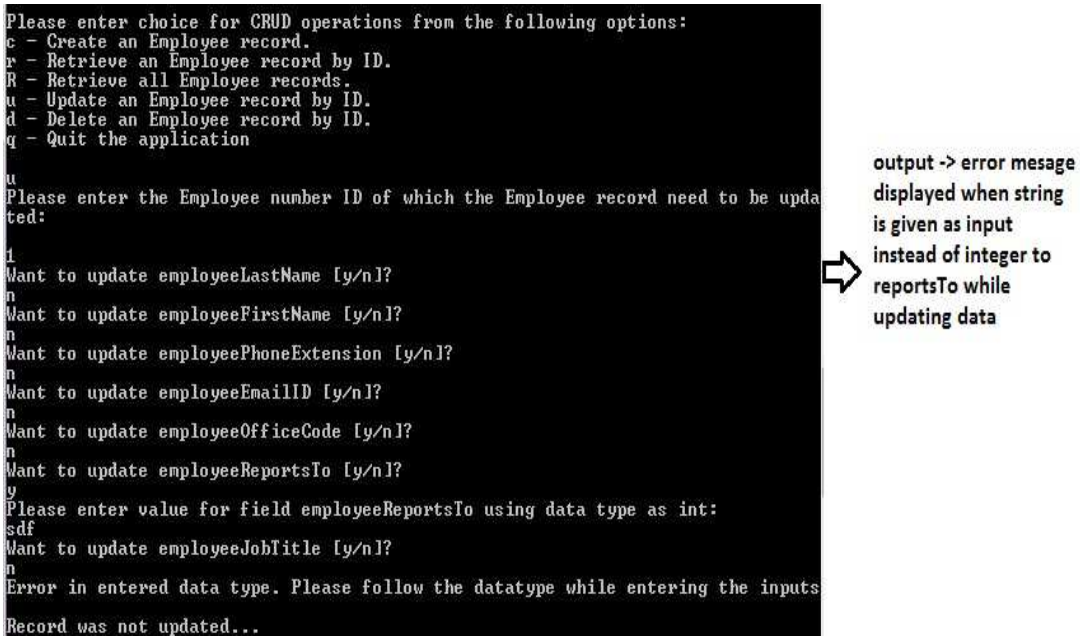
output-> update an employee record in the database when key 'u' is pressed. Asks for the employee number input and if respective record is present then asks for values of each columns of the database and updates the respective record in the database.

Snapshot 5-8:

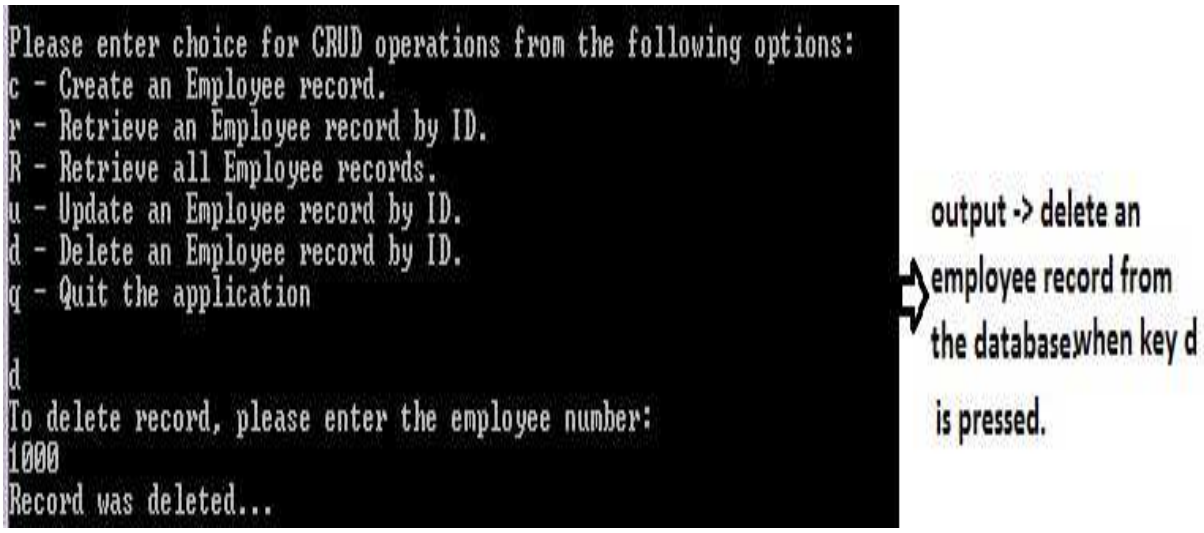


output-> error message displayed while trying to update the employee record not present in the database.

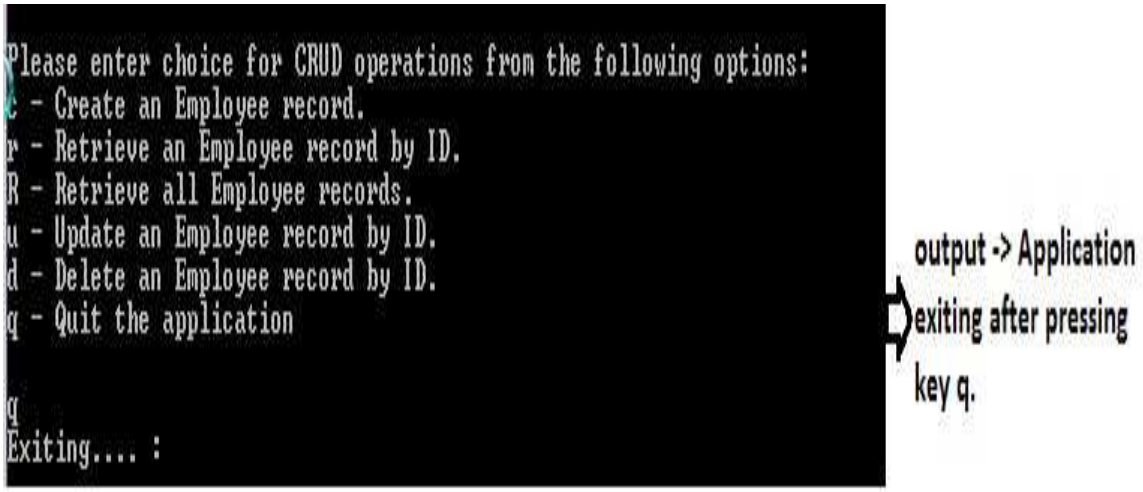
Snapshot 5-9:



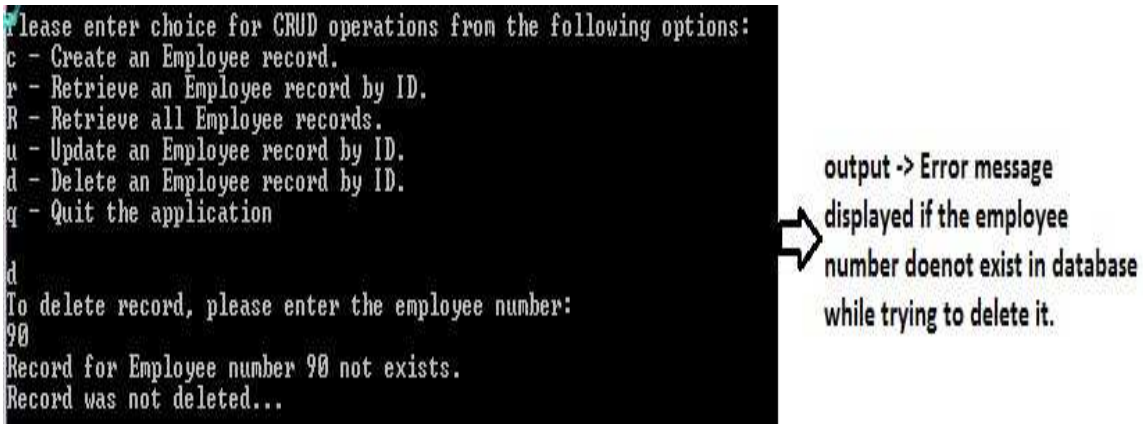
Snapshot 5-10:



Snapshot 5-11:



Snapshot 5-12:



Snapshot 5-13: