**Assignment**

OBJECT ORIENTED PROGRAMMING CSE4006

**Name:** Jayasuriya Arachchige Don Prabath Udayanga

**Student ID:** CL/HDCSE/CMU/128/14

**Module:** Object Oriented Programming - CSE4006

**Batch:** 128

**Course:**Higher Diploma in Computing and Software Engineering - CMU

**Date of Submission:** 27/07/2025

**Feedback Form**

**International College of Business & Technology**

**Module:** CSE4006 - Object Oriented Programming

**Student:**

**Assessor:**

**Assignment:** Connect Me

**Strong features of your work:**

**Areas for improvement:**

**Marks Awarded:**

**Assignment Cover Sheet**

|  |  |  |
| --- | --- | --- |
| **Qualification** | | **Module Number and Title** |
| HD in Computing and Software Engineering | | CSE4006 - **Object Oriented Programming** |
| **Student Name & No.** | | **Assessor** |
| Prabath Udayanga (CL/HDCSE/CMU/128/14) | |  |
| **Hand out date** | | **Submission Date** |
|  | | 27/07/2025 |
| **Assessment type**  WRIT1 | **Duration/Length of**  **Assessment Type**  3000 Words | **Weighting of Assessment**  100% |

|  |
| --- |
| **Learner declaration** |
| I Prabath Udayanga (CL/HDCSE/CMU/128/14) certify that the work submitted for this assignment is my own and research sources are fully acknowledged. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Marks Awarded** | | | |
| First assessor | |  | |
| IV marks | |  | |
| Agreed grade | |  | |
| Signature of the assessor |  | Date |  |

A screenshot of a computer

AI-generated content may be incorrect.

**Acknowledgment**

I would like to extend my heartfelt gratitude to Mr. Chanuka for his invaluable guidance and support throughout the Object Oriented Programming module. His clear and structured teaching, along with real-world examples and hands-on demonstrations, significantly deepened my understanding of core OOP concepts such as encapsulation, inheritance, polymorphism, and abstraction.

I truly appreciate the effort and dedication he has shown in making object-oriented principles more approachable and ensuring that we not only grasp theoretical foundations but also gain practical skills essential for modern software development.

Thank you for your encouragement and for sharing your expertise with us.

Content

Introduction.………………………………………………………………………… 5

Task 1 …………………………………………………………………......…………6

Task 2 ………………………………….……………………………………….......10

Task 3 ….…………………………………………………………………………...29

Conclusion ………………………………………….………………………………46

References ……...…...……………………………………………………………...46

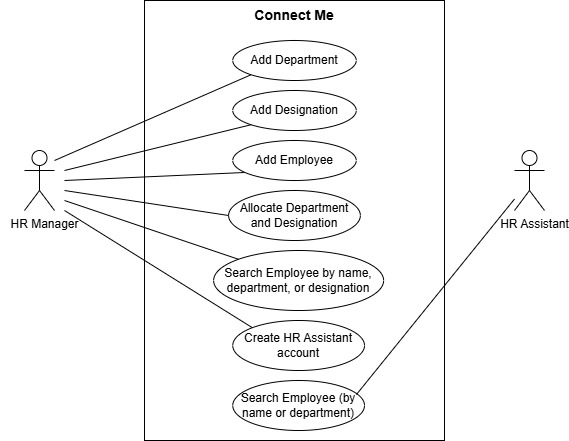
**Introduction**

Human Resource Management Systems (HRMS) is the basic digital support of the organizations that help them to administrate their workforce and to enhance their human resources activities. The proposed system employs Java to develop an automated HRMS system which processes critical data of the employees and departmental data and leave approval and pay slip generation exercises. The interface of the system is very easy and enables all the necessary HR employees functionalities and workforce needs. This project presents the effectiveness of the application of object oriented programming concepts that result to an entire HR tool that manages the performance of organizations and upholds the efficiency of data.

**Task 1: UML Diagrams**

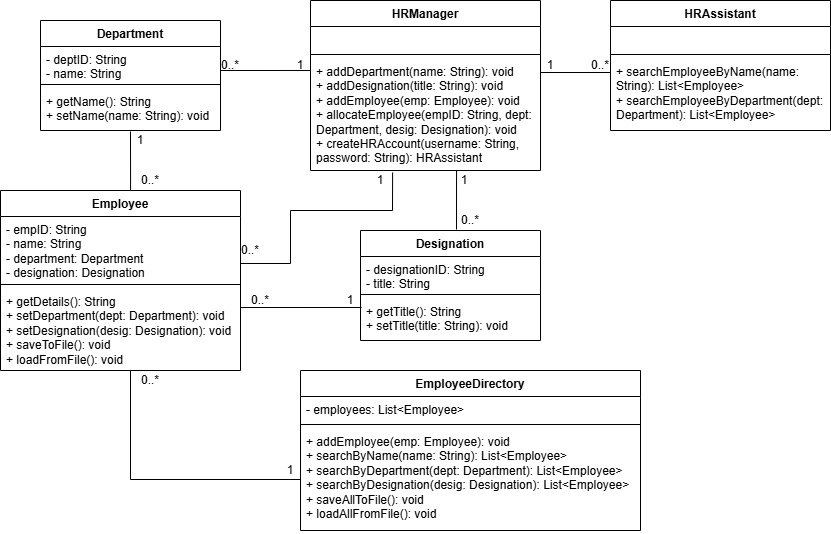
**1. Use Case Diagram**

**Actors:**

1. **HR Manager** (Primary actor)
2. **HR Assistant** (Secondary actor)

**Use Cases:**

* Within this entry, using HR Manager system, it is possible to create additional departments.
* The HR Manager plays the role of creating designations.
* The HR Manager is the person that enters this interface and inserts new employees that will have their departmental assignment and also role specification.
* The searches of the employee details can be conducted by the HR manager and by the HR assistant.
* The HR Manager will establish a system of employment account of the HR Assistant.

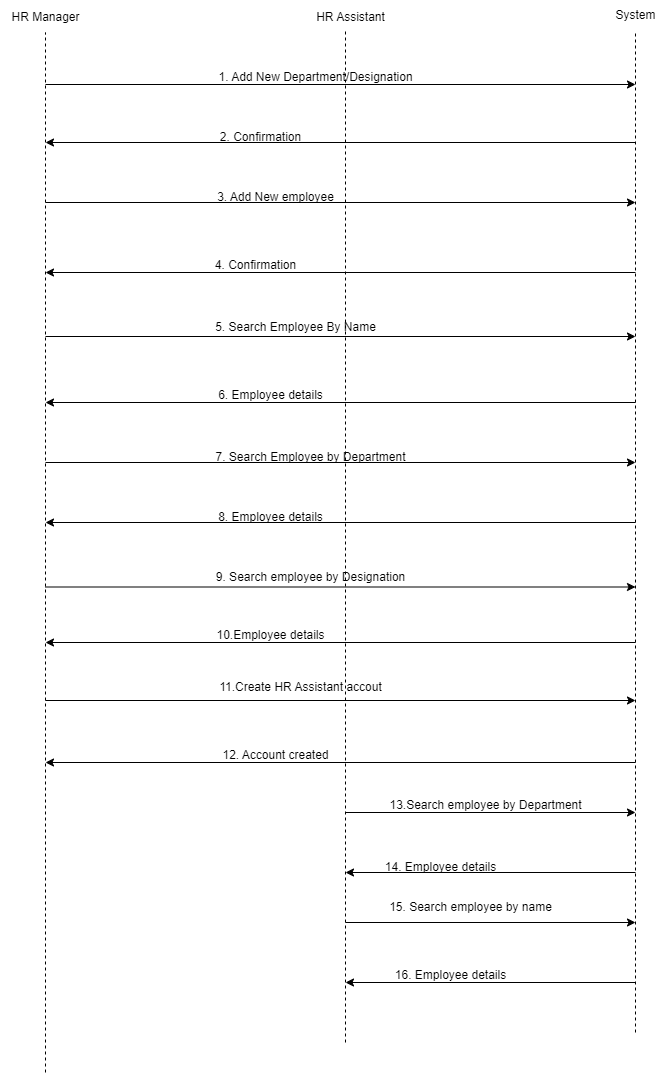
**2. Class Diagram**

1. HR creates a department: HRManager.addDepartment("Finance")
2. HR adds a job title: HRManager.addDesignation("Accountant")
3. New employee "Bob" is added and assigned:
   1. addEmployee(bob)
   2. allocateEmployee("Bob123", financeDept, accountantTitle)
4. Later, someone searches: searchEmployeeByName("Bob") → returns Bob's details.

Key Points:

* **HRManager** is the "boss" that controls everything.
* **Employees** are connected to **Departments** and **Job Titles**.
* **HRAssistant** helps with searches.
* Everything can be saved/loaded from files.

**3. Sequence Diagram**

A sequence implies how the HR Manager and the HR Assistant along with System interacts with each other to carry out the task. The diagram shows a sequence of creating an order of process in the section of departments/designations followed by adding employee then search employee criteria and creation of an account of HR Assistant. The chart shows numbered stages that show the flow of messages among the parties.

**Explanation**

**1. Departments/Designations and Employees Addition**

* The HR Manager starts the process of the creation of new departments, new appointment of the employees and the introduction of new personnel into the company.
* The subsequent of all the processes is that they are transferred to the System by the HR Manager and are handled and then the HR System informs that the processing has been completed. In such activities the HR Manager demonstrates his ability to introduce changes of definitional patterns according to the organizational structure and introduce the staff employee records to the system.

**2. Recruitment of Employees**

* The HR Assistant uses different queries of criteria to query the System to get the details of the employee

By Name: In this mode the employees can be identified through their names.

By Department: Query on employee on the basis of a specific department.

By Designation: Searching of employees with the specific designation.

* Once proper information of employees has been retrieved, the System will then display proper employee information to the HR Assistant.

**3. Setting up an HR Assistant Account**

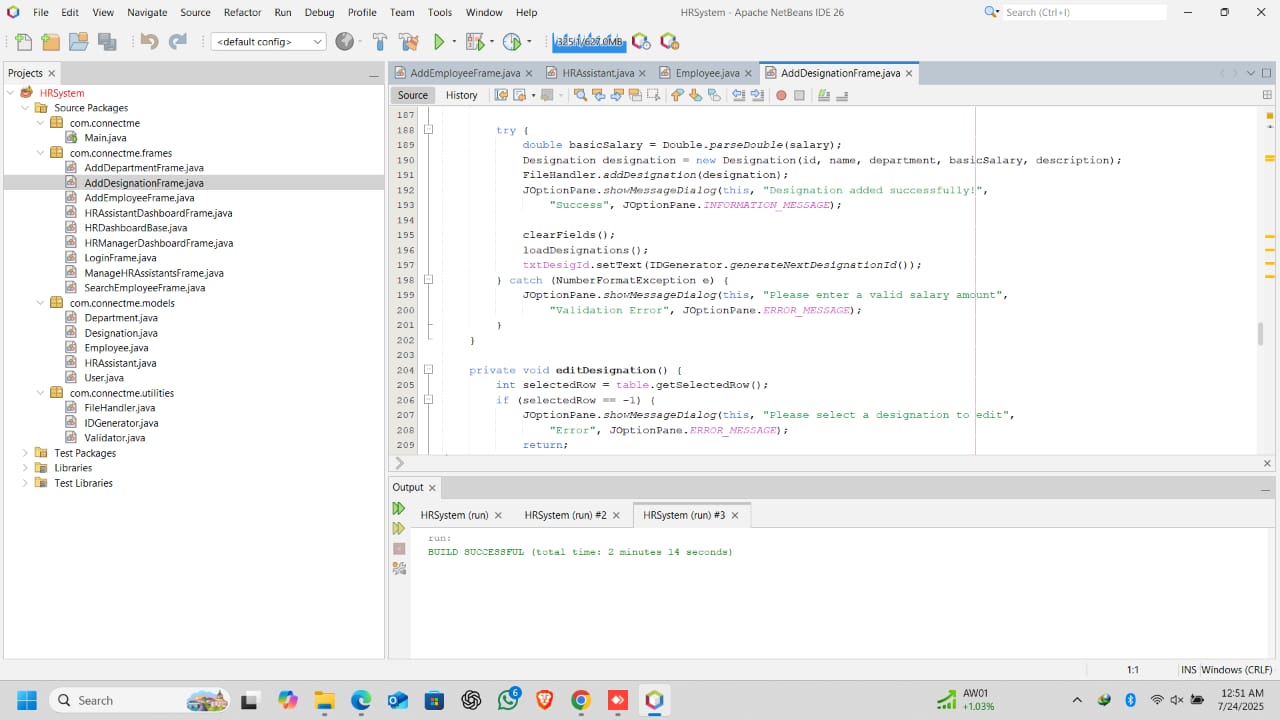
* With the system the HR Manager can be able to create new HR Assistant accounts. After this successful action of creation of an account the System will verify the account.

**4. Communication Flow**

* On the sequence diagram the decent of communications comes out between the HR Manager, HR Assistant and the System becomes visible. The senders and the recipients of a message in the drawing are represented by arrows where messages are transferred.
* The operation of tracking is made by means of the sequence numbers such that the workflow is unconfused.

**Task 2**

**1. Object**

**a) Object concept in AddDesignationFrame**

Composition of objects reflects in the two model classes (Designation and Department) with which the class works.

**Implementation:**

* Creates and manages Designation objects
* Associates designations with departments with composition
* Fills the combo box using department objects

*// Creating Designation objects*

Designation designation = new Designation(id, name, department, basicSalary, description);

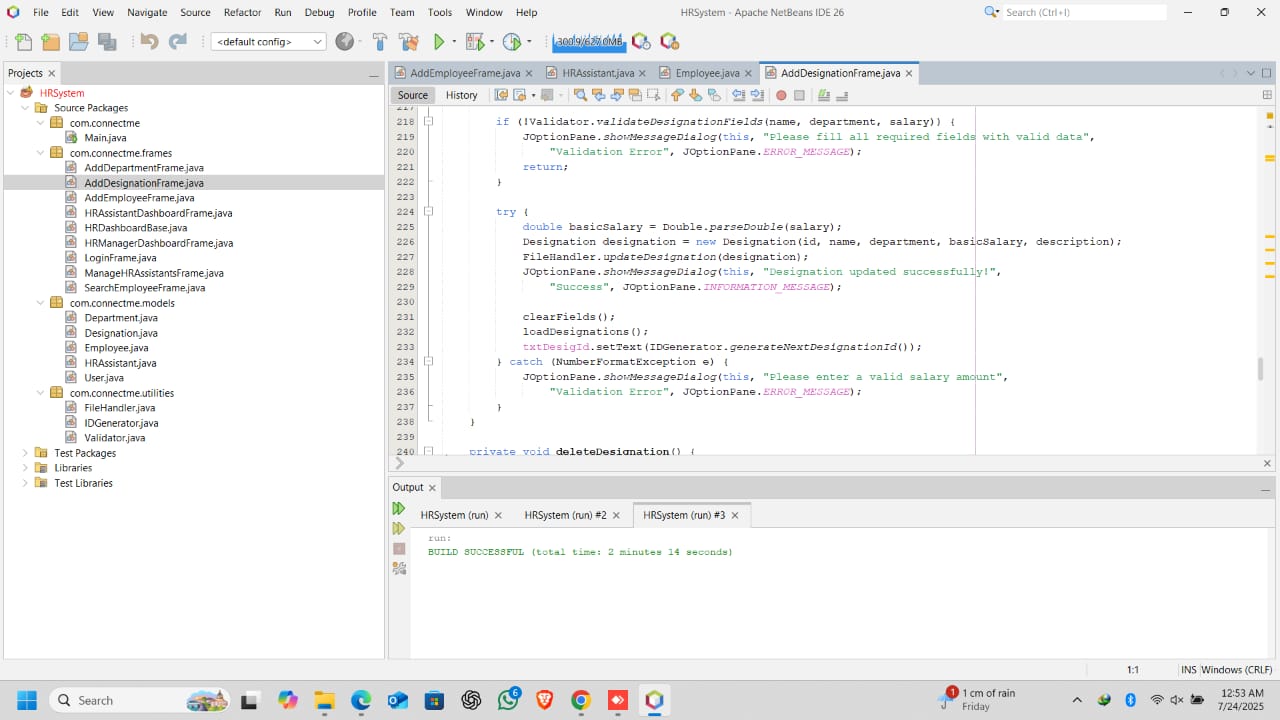
*// Using Department objects*

departments = FileHandler.readDepartments();

**Key Points:**

1. **Object Composition**:
   * In the code, it can be seen how objects of Designation are created and simultaneously a relationship is maintained with the objects of Department.
   * This is a kind of a “has-a” relationship (a Designation has a Department association).
2. **Implementation Details**:
   * The Designation constructor takes a department parameter, setting the composition.
   * The loading of departments is done in separate and to initialize UI elements (such as combo boxes).
   * This ensures that they are separated by concerns and they still enable objects to cooperate.
3. **Benefits**:
   * Models real-world relationships (designations belong to departments).
   * Permits code reuse (Department objects can be used over here).
   * Gives flexibility (dynamically can change department associations).

**2. Abstraction**

**a) Abstraction concept in AddDesignationFrame**

The frame abstracts away complex operations behind simple method calls.

**Implementation:**

* Hides file operations behind FileHandler methods
* Abstracts department loading and name resolution
* Validation logic is built to Validator

*// Abstracting file operations*

FileHandler.addDesignation(designation);

FileHandler.updateDesignation(designation);

*// Abstracting department name resolution*

String departmentName = getDepartmentName(desig.getDepartmentId());

**Key Aspects of Abstraction**

1. **Hiding Implementation Details**  
   The frame:

* Doesn't need to know *how* files are saved/loaded
* Doesn't care *how* department names are resolved
* Doesn't handle validation rules directly

1. **Three Levels of Abstraction Shown:**

*// 1. File Operations Abstraction*

FileHandler.addDesignation(designation); *// Hides file I/O complexity*

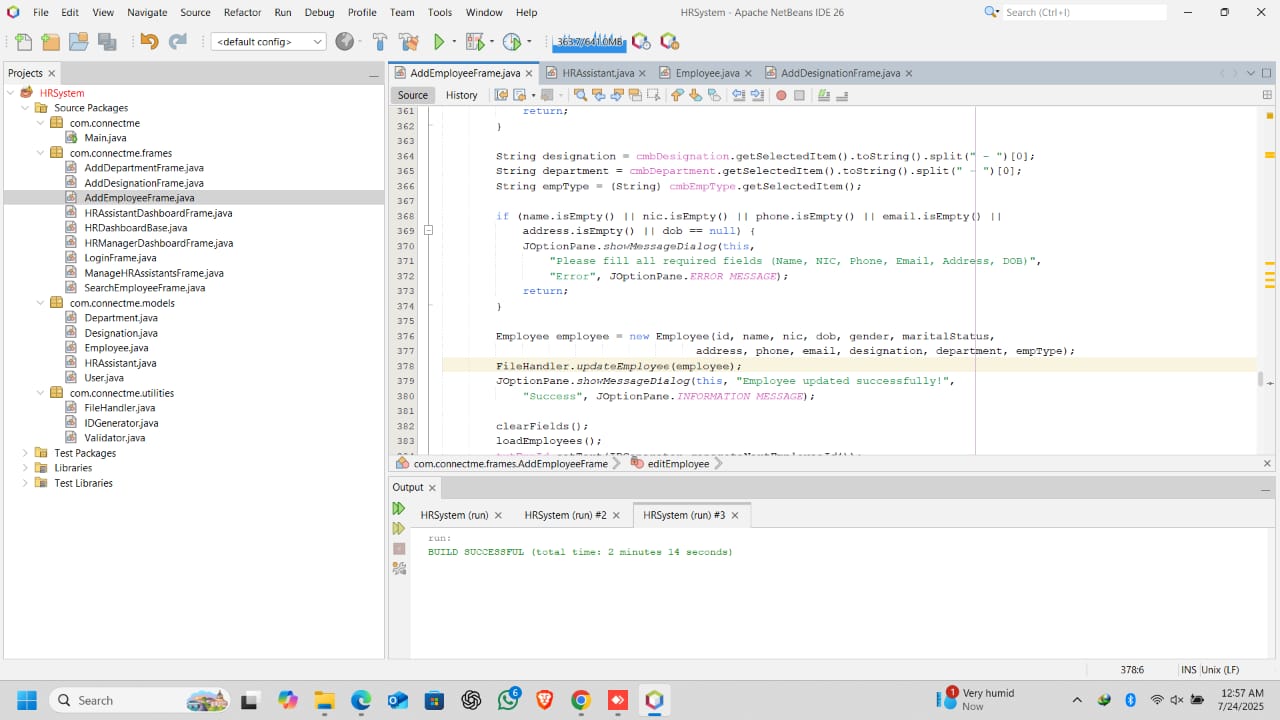
*// 2. Relationship Resolution Abstraction*

String deptName = getDepartmentName(designation.getDepartmentId()); *// Hides ID lookup logic*

*// 3. Validation Abstraction*

Validator.validateDesignationFields(...); *// Hides regex/rule checks*

1. **Real-World Analog**:  
   Such instance of driving a car - you drive using the steering wheel (a simple interface) and you do not need to know the adaptation of the power steering (a complex implementation).

**b)** **Abstraction concept in AddEmployeeFrame**

The frame abstracts complex operations behind simple interfaces.

**Implementation:**

* Hides file operations behind FileHandler
* Abstracts date handling with JDateChooser
* Delegates validation to Validator
* Manages complex employee-department-designation relationships

*// Abstracting file operations*

FileHandler.addEmployee(employee);

FileHandler.updateEmployee(employee);

*// Abstracting relationship resolution*

String designationName = getDesignationName(emp.getDesignationId());

String departmentName = getDepartmentName(emp.getDepartmentId());

**Abstraction in the Employee Management Frame**

This is an example of how complicated HR processes can be reduced with a layered abstraction:

**1. File Operations Abstraction**

FileHandler.addEmployee(employee); *// What you see*

*Hidden Complexity*:

* File paths
* Data serialization
* Error handling
* Concurrent access management

**2. Relationship Resolution Abstraction**

String deptName = getDepartmentName(emp.getDepartmentId());

*Hidden Complexity*:

* Database/File lookups
* ID-to-object mapping
* Caching mechanisms
* Null reference checks

**3. Validation Abstraction**

Validator.validateEmployeeFields(...);

*Hidden Complexity*:

* Regular expressions
* Business rules
* Data type conversions
* Cross-field validation logic

**4. UI Component Abstraction**

JDateChooser datePicker; *// What you use*

*Hidden Complexity*:

* Calendar rendering
* Localization
* Date formatting
* Event handling

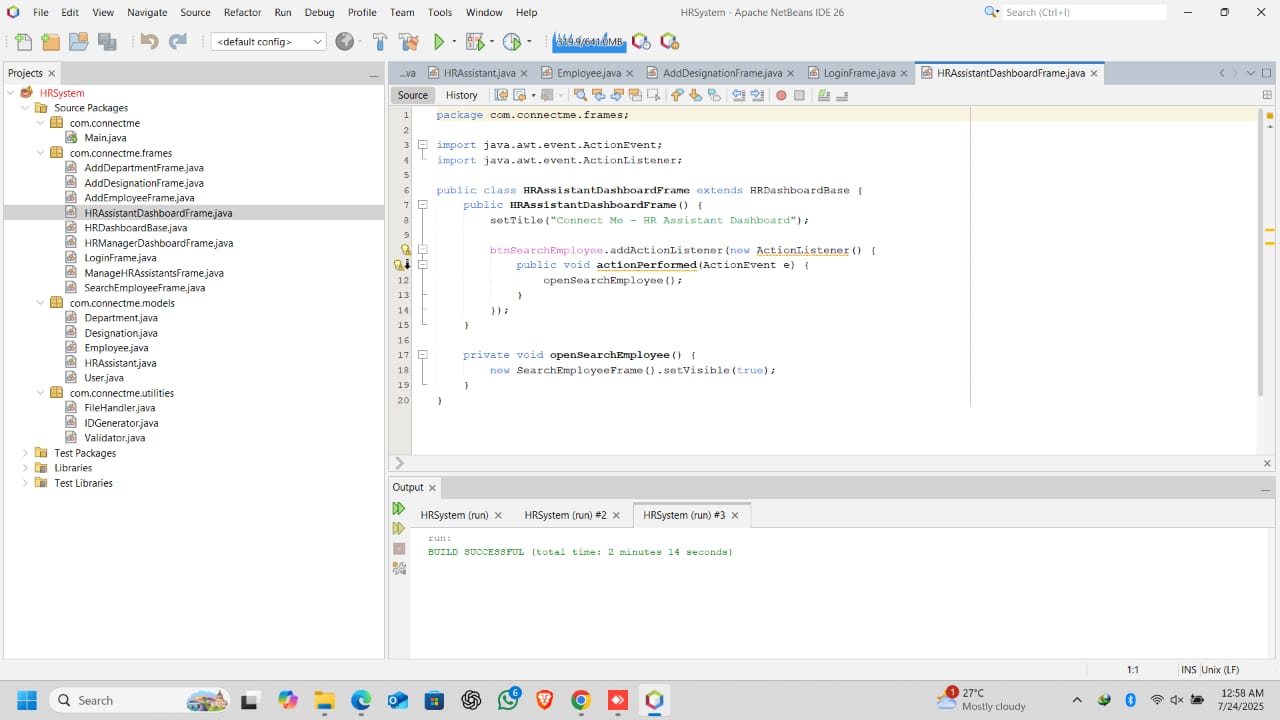
**Key Abstraction Benefits**

1. **Focus Isolation**
   * UI developers work with simple method calls
   * Business logic developers maintain validation rules
   * Data engineers optimize storage separately
2. **Change Resilience**
   * Switch from file storage to database by modifying only FileHandler
   * Update validation rules without touching UI code
   * Replace date picker widget without breaking employee creation logic
3. **Debugging Efficiency**
   * Errors are contained within abstracted components
   * Stack traces lead directly to responsible layers

**Real-World Parallel**  
Like an HR manager using a software form:

* They enter employee data (simple interface)
* Don't need to know how:
  + Data is saved (storage abstraction)
  + Dates are validated (validation abstraction)
  + Departments are resolved (relationship abstraction)

**3) Inheritance**

**a)** **Inheritance concept in HRAssistantDashboardFrame**

This is the brightest OOP idea that is shown in this class.

**Implementation:**

* Extends HRDashboardBase class
* Inherits all functionality from the base class
* Can override or extend base class behavior

public class HRAssistantDashboardFrame extends HRDashboardBase {

*// Inherits all members from HRDashboardBase*

}

This frame demonstrates abstraction through inheritance from HRDashboardBase:

**1. Inherited Abstraction**

public class HRAssistantDashboardFrame extends HRDashboardBase

*What's Hidden*:

* All base dashboard setup code (window sizing, panel creation)
* Common UI components (buttons, layouts)
* Default event handlers

**2. Concrete Implementation**  
While inheriting core functionality, the frame:

* **Specializes** for HR Assistant role
* **Extends** with assistant-specific features
* Potentially **overrides** base methods

**3. Abstraction Benefits**

*// Simple interface for derived frames:*

setTitle("HR Assistant Dashboard"); *// Just configure your specialization*

*Hidden Complexity*:

* Parent class handles:
  + Window lifecycle management
  + Common UI boilerplate
  + Standard event handling

**Key OOP Principles Demonstrated**:

1. **Layered Abstraction**  
   Base class abstracts dashboard fundamentals → Child focuses on role-specific needs
2. **Template Method Pattern**  
   Base class defines structure → Child fills in specifics
3. **Polymorphism**  
   Can treat all dashboards as HRDashboardBase type

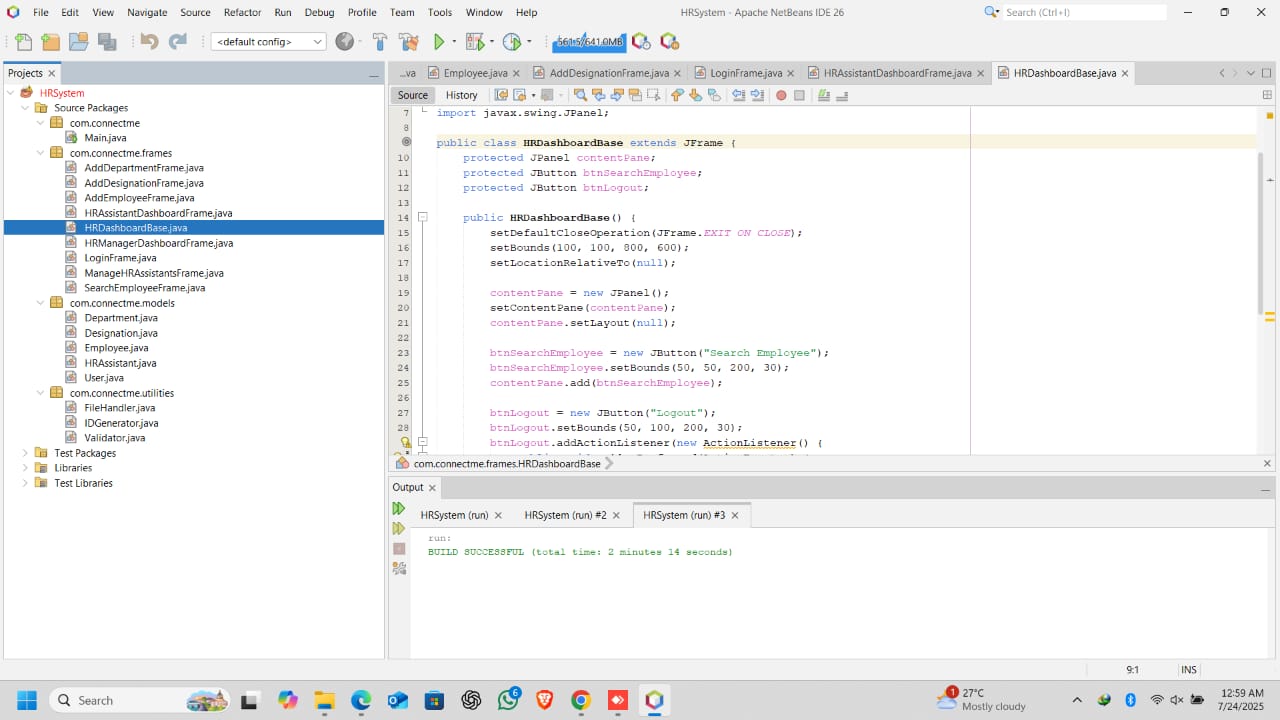
**Real-World Analogy**:  
Like a company's standard workstation setup:

* IT provides preconfigured base computer (abstracts setup complexity)
* HR Assistant just adds their specialized software (concrete customization)

**Why This Matters**:

* **Efficiency**: Avoid duplicating dashboard boilerplate
* **Consistency**: All dashboards share core behavior
* **Security**: Base class enforces common security checks
* **Maintainability**: Update base features once for all dashboards

It is inheritance abstraction - the child frame works with a simplified interface and the parent deals with complex common operations.

**b) Inheritance concept in HRDashboardBase**

The HRDashboardBase class illustrates the few core concepts of OOP concepts on which the specialized dashboards stand.

1. **Inheritance (Base Class)**

The class resembles a parent (base) class of specialized dashboards, which is a JFrame inheritor.

**Implementation:**

* Extends JFrame to inherit all Swing frame functionality
* Designed to be extended by classes like HRAssistantDashboardFrame

public class HRDashboardBase extends JFrame {

*// Base functionality for all HR dashboards*

}

**Inheritance Hierarchy in Action**

This base class creates a two-level inheritance structure,

1. **First-Level Inheritance** (Framework Level)

public class HRDashboardBase extends JFrame

* Gains all core window capabilities from Swing's JFrame:
  + Window management (open/close/minimize)
  + Event handling infrastructure
  + Layout and rendering functionality

1. **Second-Level Inheritance** (Application Level)

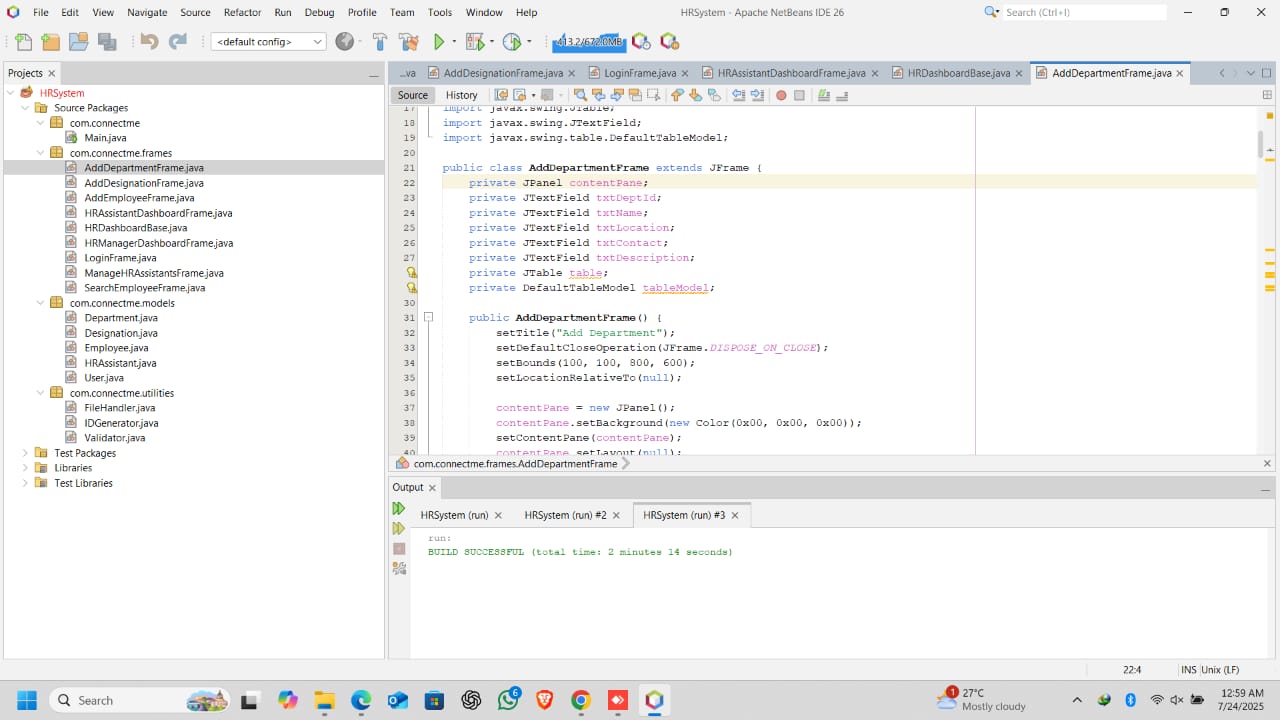
public class HRAssistantDashboardFrame extends HRDashboardBase

* Inherits both:
  + Swing framework features (via JFrame)
  + Custom HR dashboard features (via HRDashboardBase)

**Key Benefits**:

1. **Code Reuse**
   * Automatically, all dashboards develop right window behavior
   * Common UI elements (logout button, search panel) defined once
2. **Consistent Behavior**
   * Standardized window closing/opening
   * Uniform look-and-feel across all dashboards

**4) Encapsulation**

**a) Encapsulation concept in AddDepartmentFrame**

Encapsulation is the bundling of data with the methods that operate on that data, and restricting direct access to some of an object's components.

**Implementation:**

* All fields are declared as private (private JTextField txtDeptId, etc.)
* Access to these fields is controlled through public methods
* The internal state of the frame is protected from direct external manipulation

private JTextField txtDeptId; *// Private field*

private JTextField txtName;

*// ... other private fields*

*// Public methods to interact with these fields*

public void addDepartment() { ... }

public void editDepartment() { ... }

This frame protects its internal state through:

**1. Field Protection**

private JTextField txtDeptId; *// Private UI components*

private JTextField txtName; *// Cannot be accessed directly from outside*

**2. Controlled Access**

public void addDepartment() {

*// The ONLY way to add a department*

String id = txtDeptId.getText(); *// Internal access allowed*

String name = txtName.getText();

*// Validation and processing...*

}

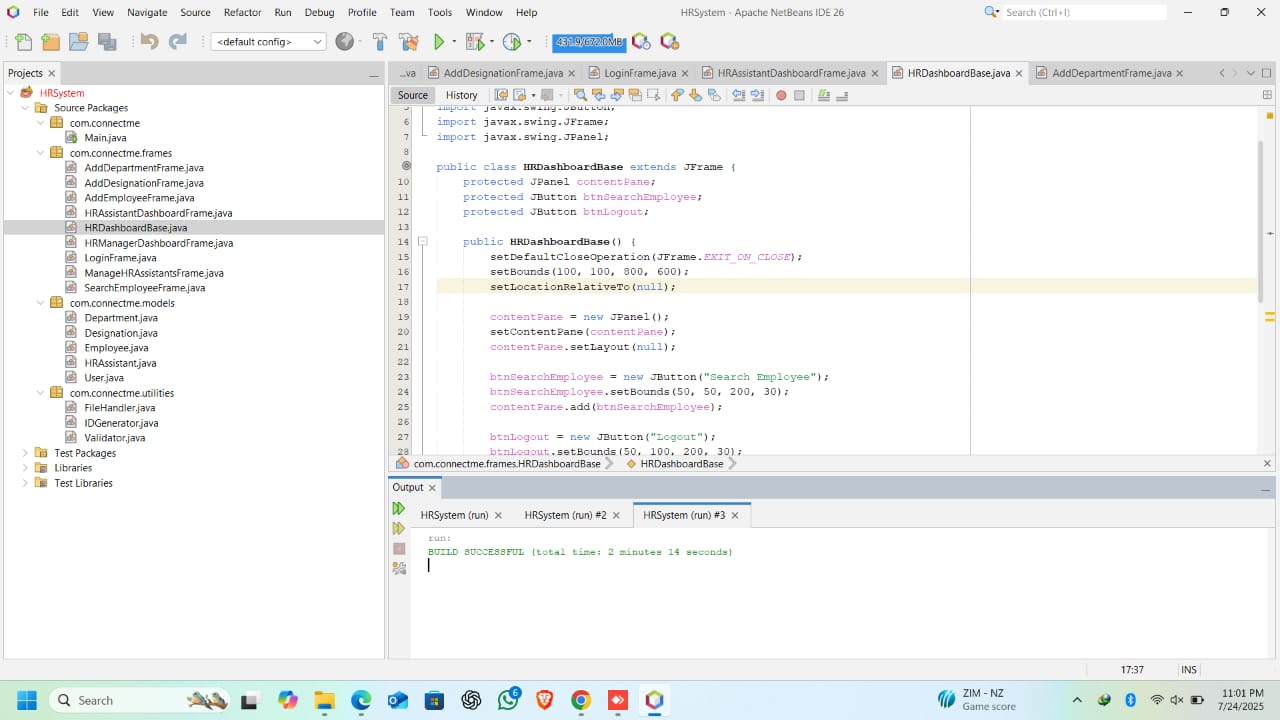
**3. Why This Matters**:

* **Input Control**: Ensures data is validated before processing  
  (e.g., prevents blank department names)
* **Change Safety**: Can modify text field types without breaking other classes  
  (e.g., switch JTextField to JFormattedTextField)
* **State Consistency**: Prevents invalid UI states  
  (e.g., stops external code from clearing fields mid-operation)

**Real-World Analogy**:  
Like an HR department's filing cabinet:

* Files (data) are stored in locked drawers (private fields)
* Only authorized staff (public methods) can:
  + Add files (addDepartment())
  + Edit files (editDepartment())
  + With proper paperwork (validation)

**Key Benefit**: The frame gives complete control over how its data is accessed and modified, so it prevents corruption, misuse and henceforth maintains the data integrity.

**b) Encapsulation concept in HRDashboardBase**

Protected access modifiers are referred to as a way of granting subclass access but limiting access to the public.

**Implementation:**

* UI components declared as protected (not private)
* Protected logout method allows subclass customization
* Hides implementation details from external classes

protected JPanel contentPane;

protected JButton btnSearchEmployee;

protected JButton btnLogout;

protected void logout() { ... }

**Strategic Access Control**

This base class uses protected modifiers to create a carefully balanced encapsulation:

**1. Protected Components**

protected JPanel contentPane; *// Accessible to subclasses but hidden from outsiders*

protected JButton btnLogout;

* *Why?* Lets child dashboards:
  + Add components to the panel
  + Modify button properties
  + While still preventing arbitrary external access

**2. Protected Method**

protected void logout() {

this.dispose();

new LoginFrame().setVisible(true);

}

* *Design Intent*:
  + Subclasses can override logout behavior
  + External classes can only call public methods
  + Base functionality remains consistent

**3. Why This Matters**:

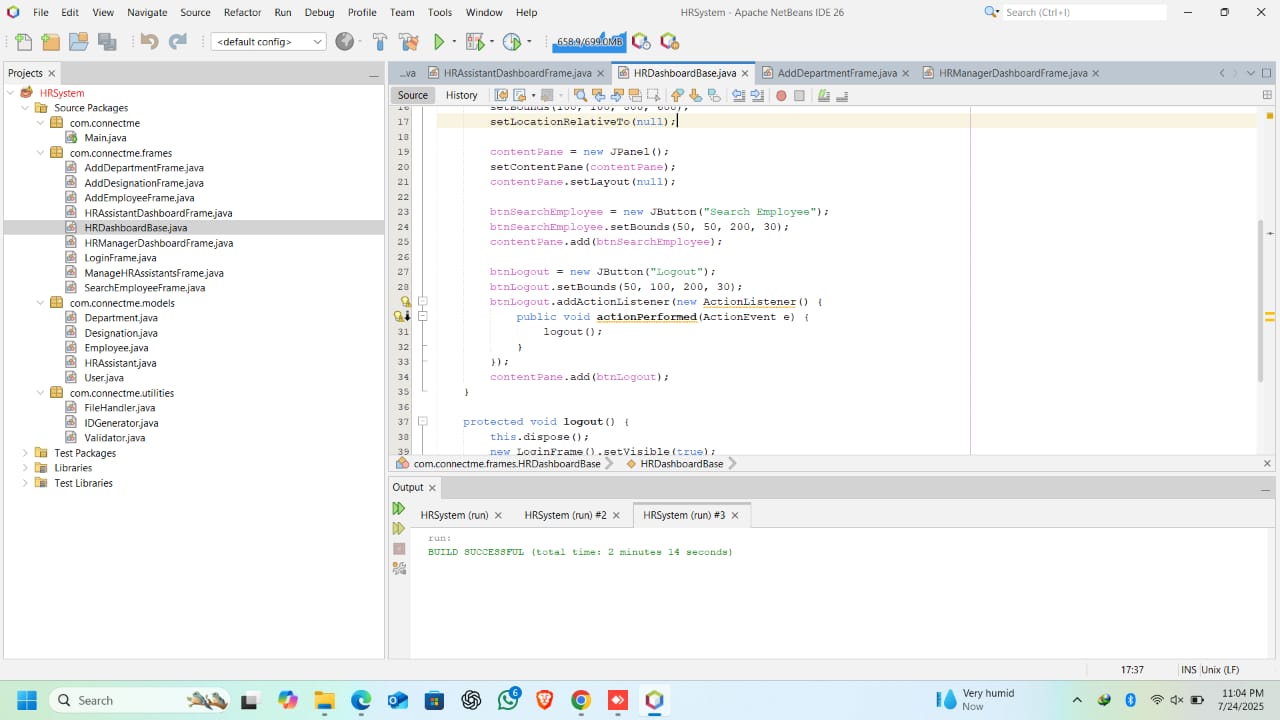
* **Framework Flexibility**: Child classes get "just enough" access to customize
* **Security**: Critical operations (like logout) can't be triggered arbitrarily
* **Maintainability**:
  + Can change UI structure in base class
  + Without breaking existing subclasses
  + While preventing misuse by other code

**Real-World Analogy**:  
Like a parent company's HR system:

* Branch offices (subclasses) get access to templates
* Can customize certain elements
* But can't break core security protocols

**Key Takeaway**: This is encapsulation with inheritance consideration - more restrictive than public and more open than private and they made a sort of half-way kind of thing labeled a "trusted family" level of access.

**5)** **Polymorphism**

**a) Polymorphism concept in HRDashboardBase**

Polymorphism is employed by means of interface realizations and way of overriding.

**Implementation:**

* ActionListener implementations for button actions
* Overriding DefaultTableModel behavior
* Using JComboBox with a custom model

*// Polymorphic ActionListener*

btnAdd.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

addDesignation();

}

});

*// Custom table model behavior*

tableModel = new DefaultTableModel(columnNames, 0) {

@Override

public boolean isCellEditable(int row, int column) {

return false;

}

};

These three polymorphism approaches are shown for this base class,

**1. Interface Implementation (ActionListener)**

btnAdd.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

addDestination(); *// Runtime binding to specific action*

}

});

* *What's Polymorphic*: Same actionPerformed method behaves differently for each button
* *Benefit*: Decouples button wiring from action logic

**2. Method Overriding (TableModel)**

tableModel = new DefaultTableModel(...) {

@Override

public boolean isCellEditable(...) {

return false; *// Custom behavior*

}

};

* *What's Polymorphic*: Same method call (isCellEditable) now has modified behavior
* *Benefit*: Changes table behavior without creating full custom class

**3. Generic Component Usage (JComboBox)**

JComboBox<String> cmbRoles = new JComboBox<>();

* *What's Polymorphic*: Treats all combo boxes as JComboBox regardless of content type
* *Benefit*: Can swap data models without changing UI code

**Key Polymorphism Benefits Here**:

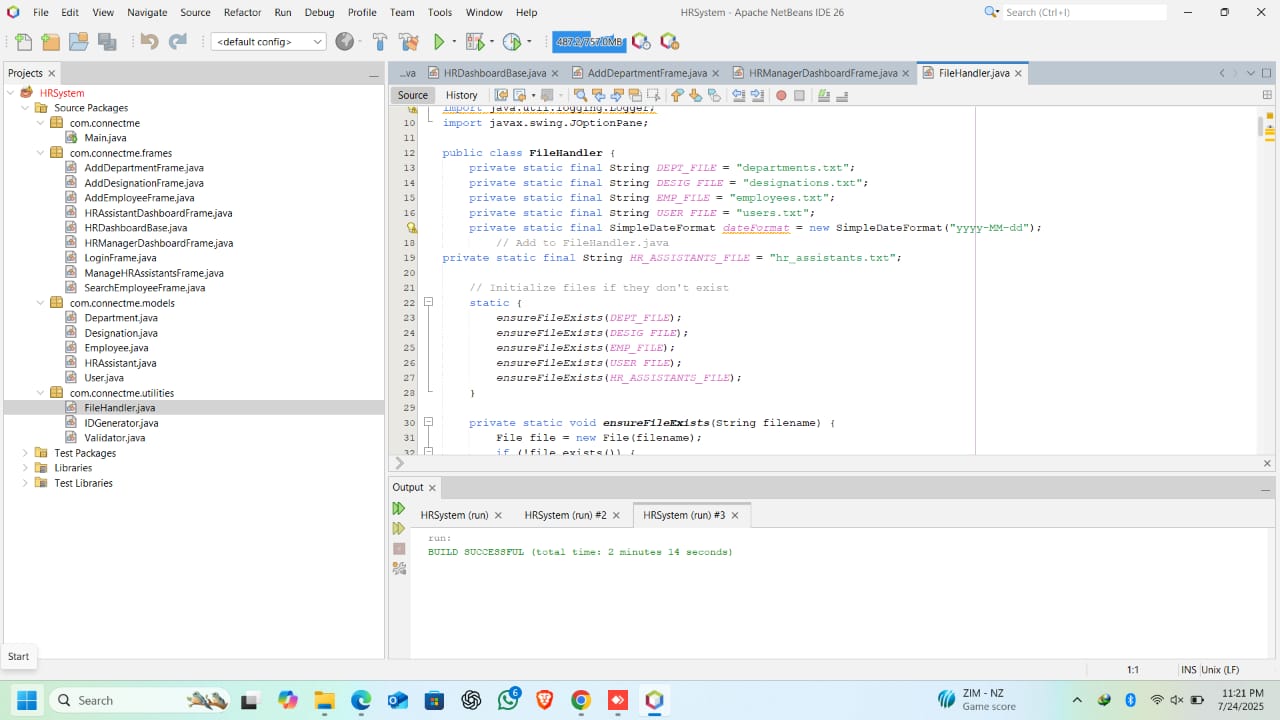
1. **Extensibility**: New button actions can be added without modifying base class
2. **Customization**: Child classes can override table behavior
3. **Type Flexibility**: UI components work with different data types

**Real-World Analogy**:  
Like an HR hotline button panel:

* Same "press" action (polymorphic interface)
* Does different things for:
  + "Benefits" button → Shows benefits info
  + "Payroll" button → Shows payroll forms
* While using the same physical interface

Such a polymorphic design not only makes dashboard framework flexible but also maintains it.

**6) Class**

**a) Class concept in FileHandler**

A class defines:

* **Data (Attributes/State)**:

private static final String DEPT\_FILE = "departments.txt"; *// Class-level data*

* **Behavior (Methods)**:

public static void addDepartment(Department d) { ... } *// Class-level behavior*

* **Class Purpose**:  
  A blueprint for **file operations**, grouping related methods/constants.

public class FileHandler { *// Class declaration*

*// Data (constants)*

private static final String DEPT\_FILE = "...";

*// Behavior (methods)*

public static void addDepartment(Department d) { ... }

}

**Task 3: User Manual**

**Introduction**

The Connect Me HR Management System is an application which is aimed at automatization the work with employees, permitting the HR Managers to create departments, add designations, add employees and create accounts for HR Assistant. Employee details can be searched easily by the HR Assistants by logging into the system.

**System Requirements**

* **Hardware:** Minimum 4GB RAM, 1.5 GHz Processor
* **Software:** Windows 10 or 11, Java Runtime Environment

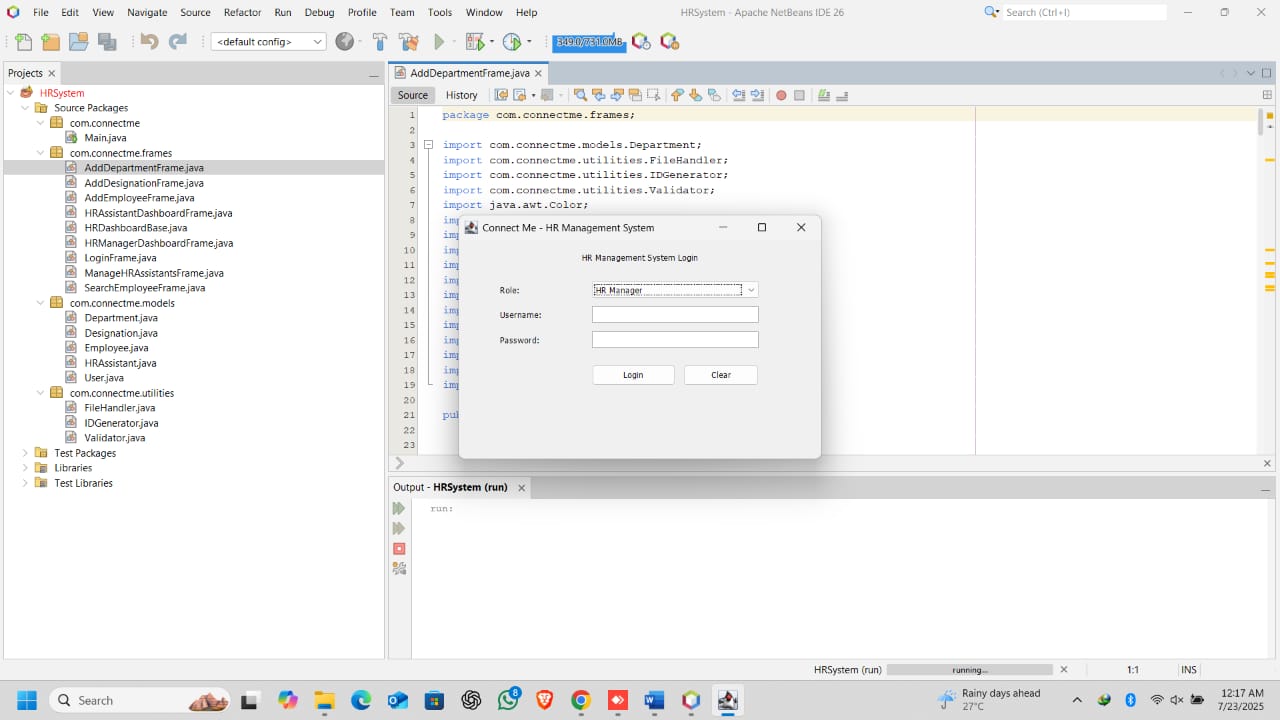
**How to Install**

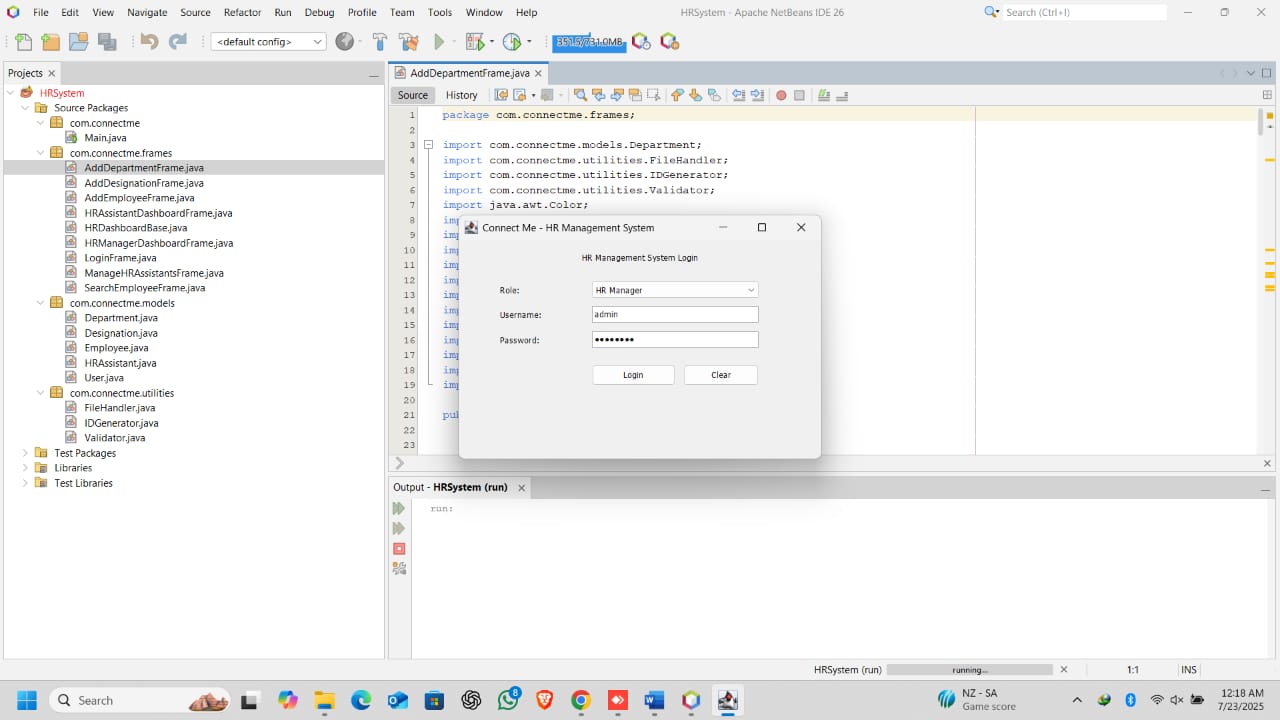
* 1. Install **Java** and **NetBeans**
  2. **Open** the project (Browse to the folder where the Connect Me HR Management System project is saved.)
  3. Click **main** **class** and **Run** the file.

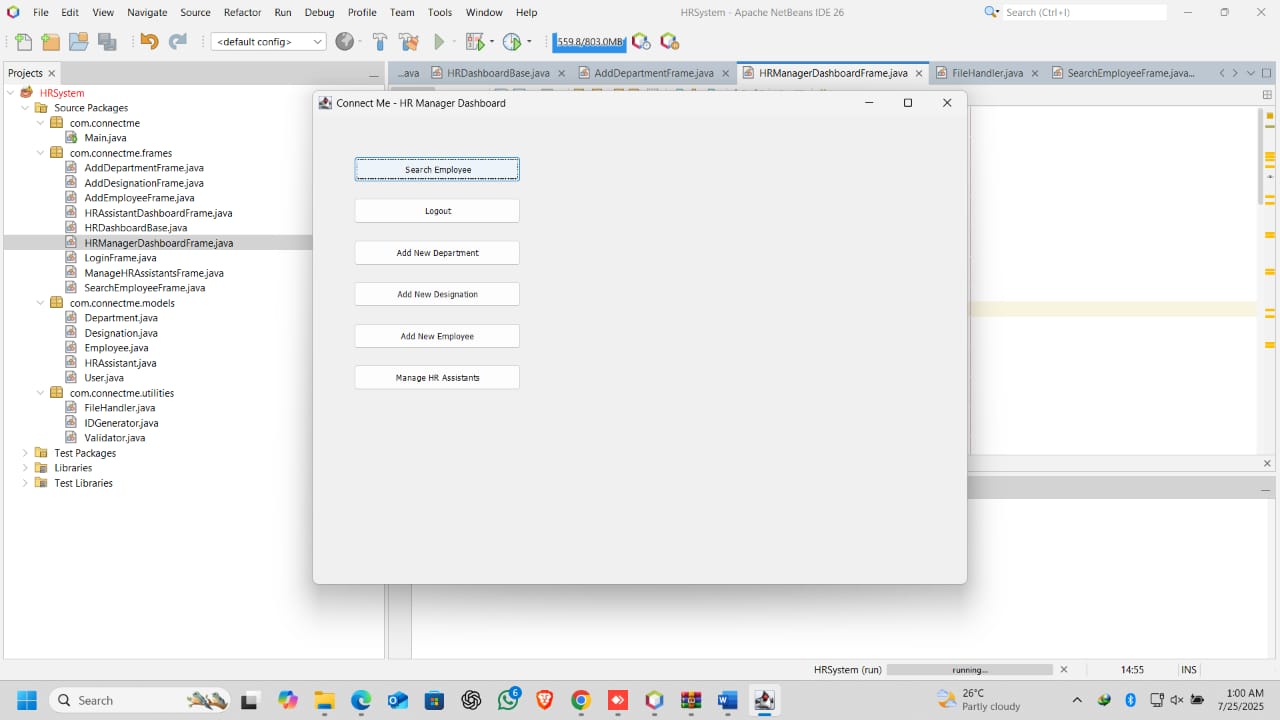
The program will be launched successfully!

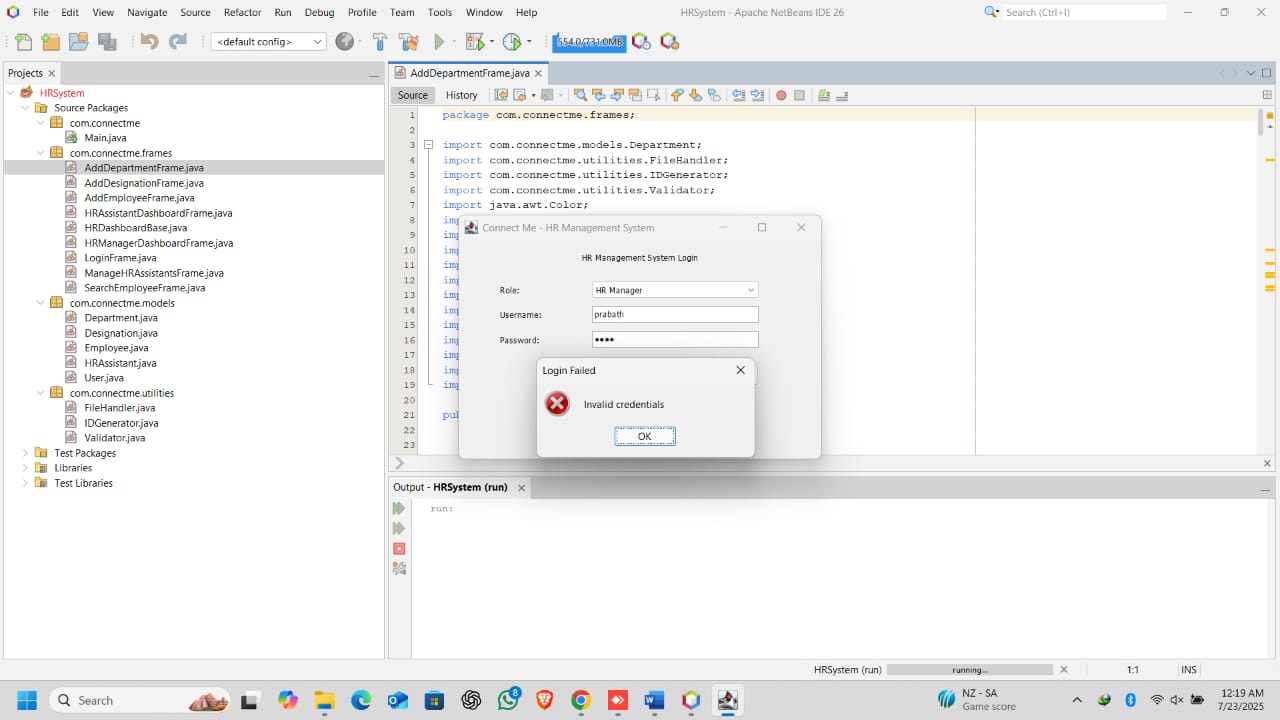
**The Login Instructions**

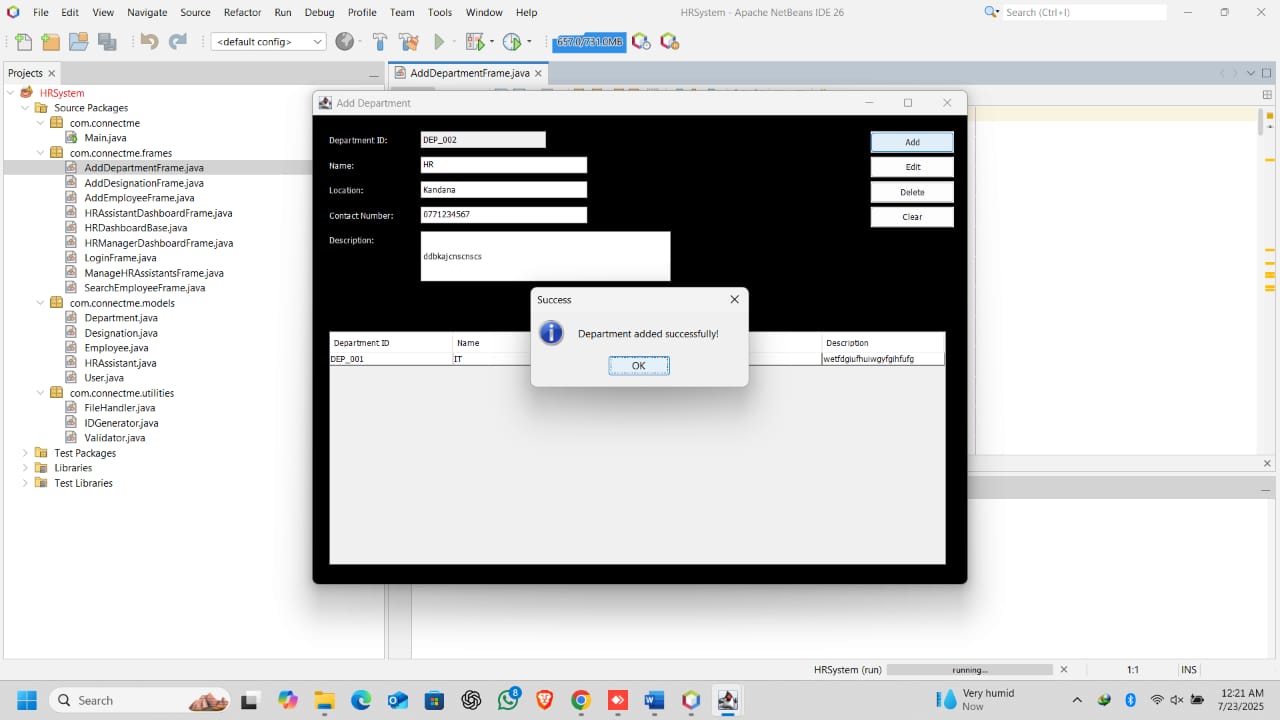
* 1. Open the program.
  2. Select HR Manager option.
  3. Enter the username (**admin**) and password (**admin123**).
  4. Click **Login** button.
  5. HR Manager dashboard opens.

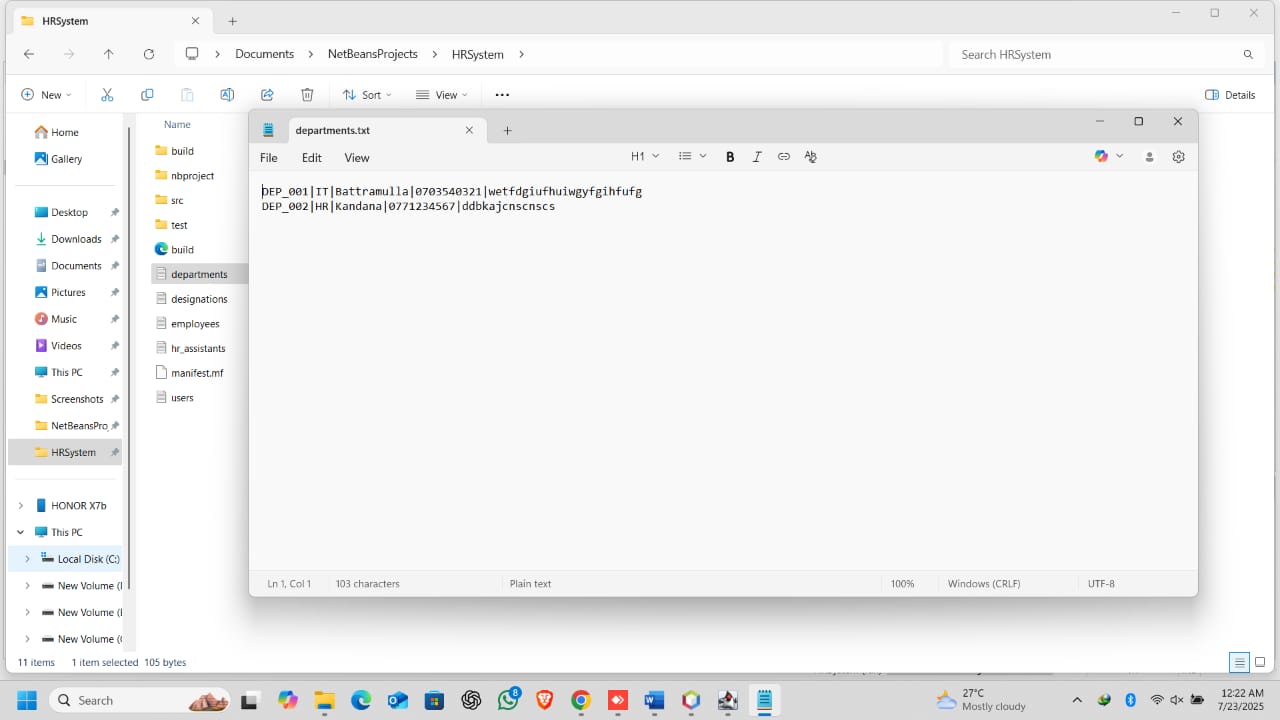
**1. Main login**

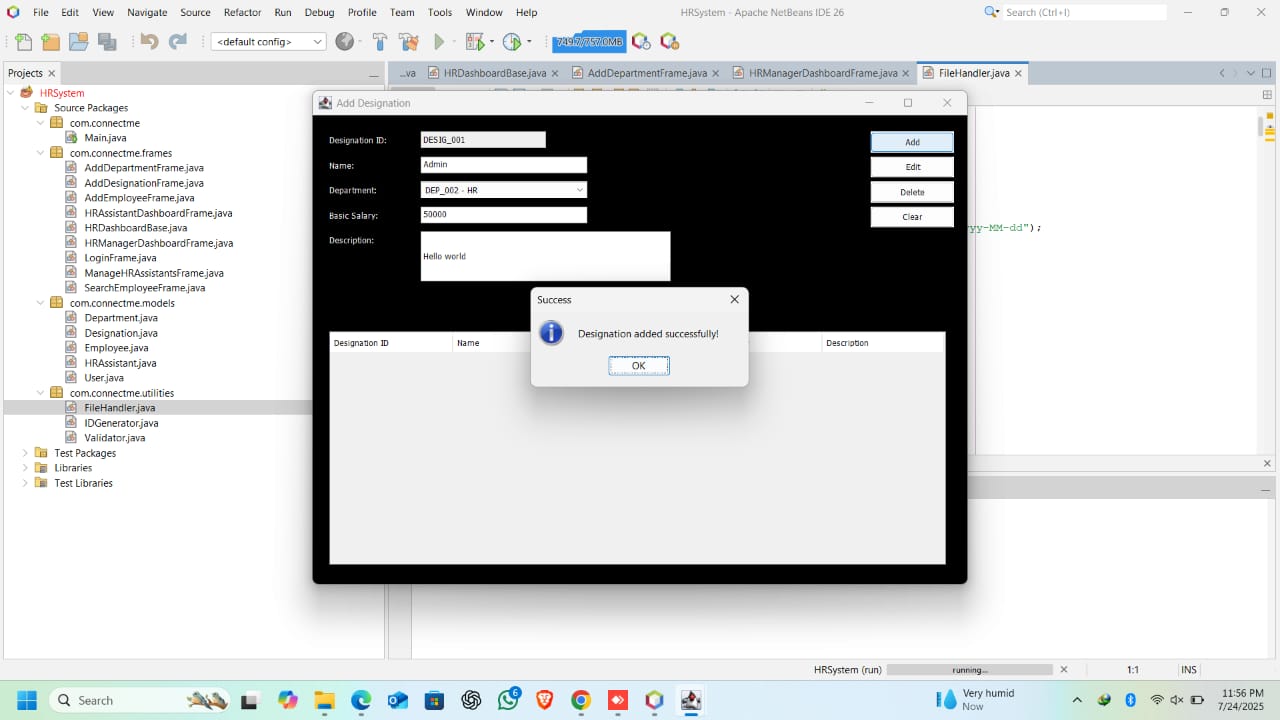
**2. Manager should login using username as “admin” password as “admin123”. The manager can log into the Manager frame.**

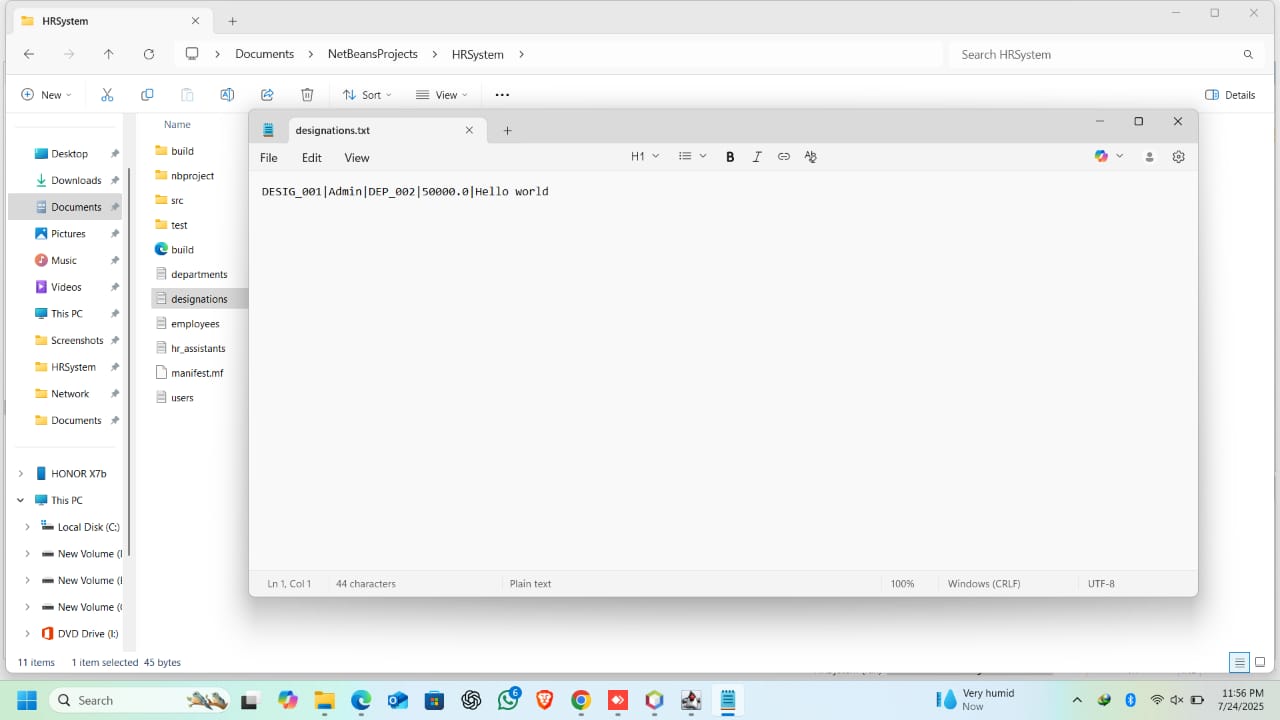
****

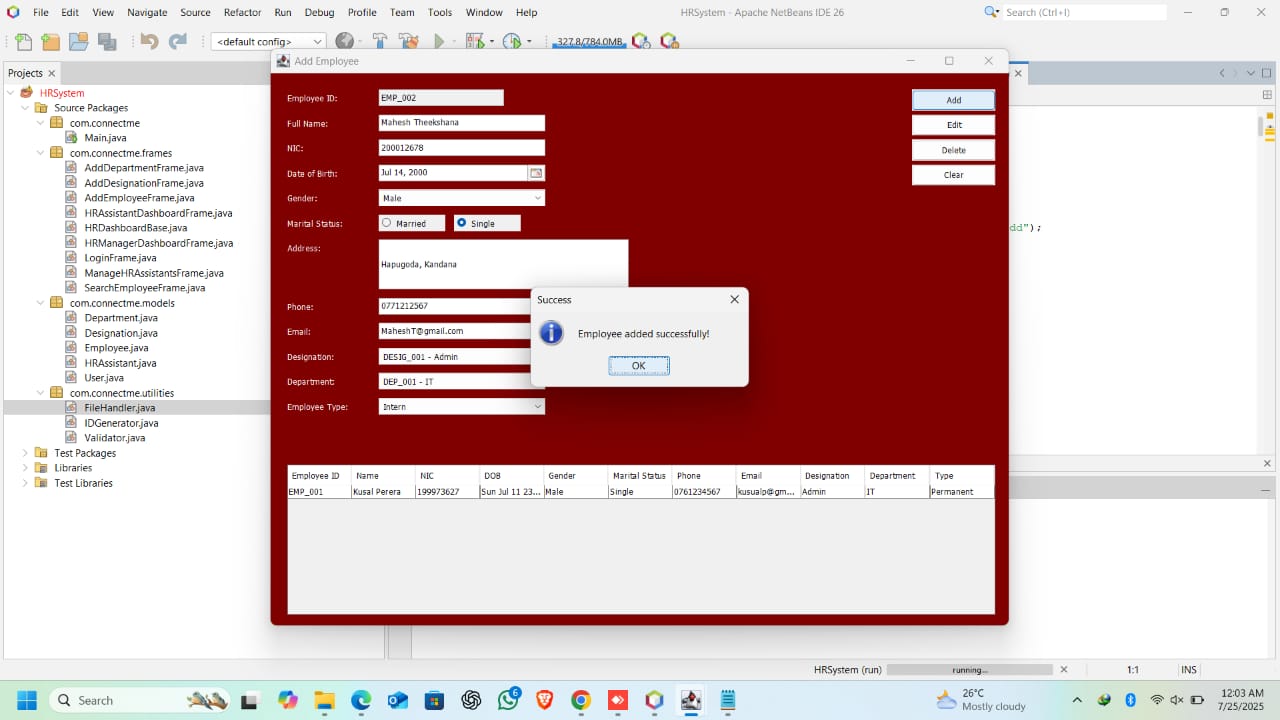
**3. If the username or the Password is wrong, an error message will display.**

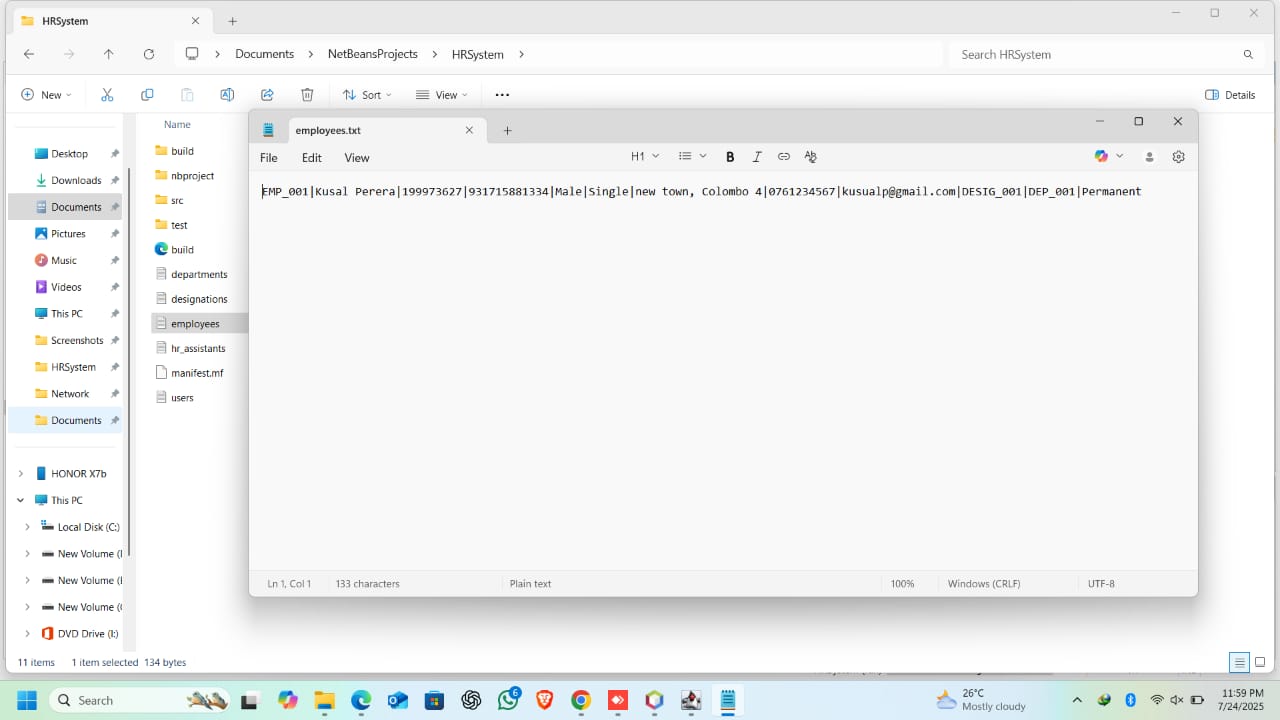
**4. Manager can Add departments successfully and the data will be stored in “departments” File.**

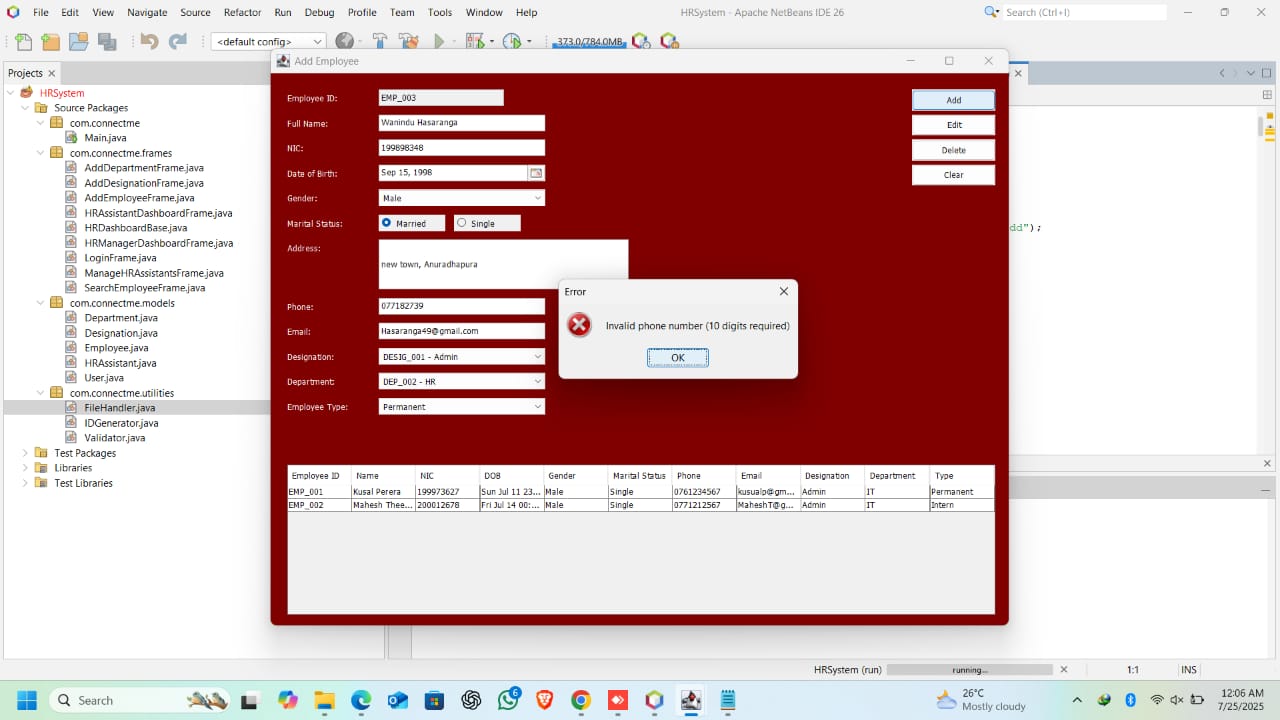
****

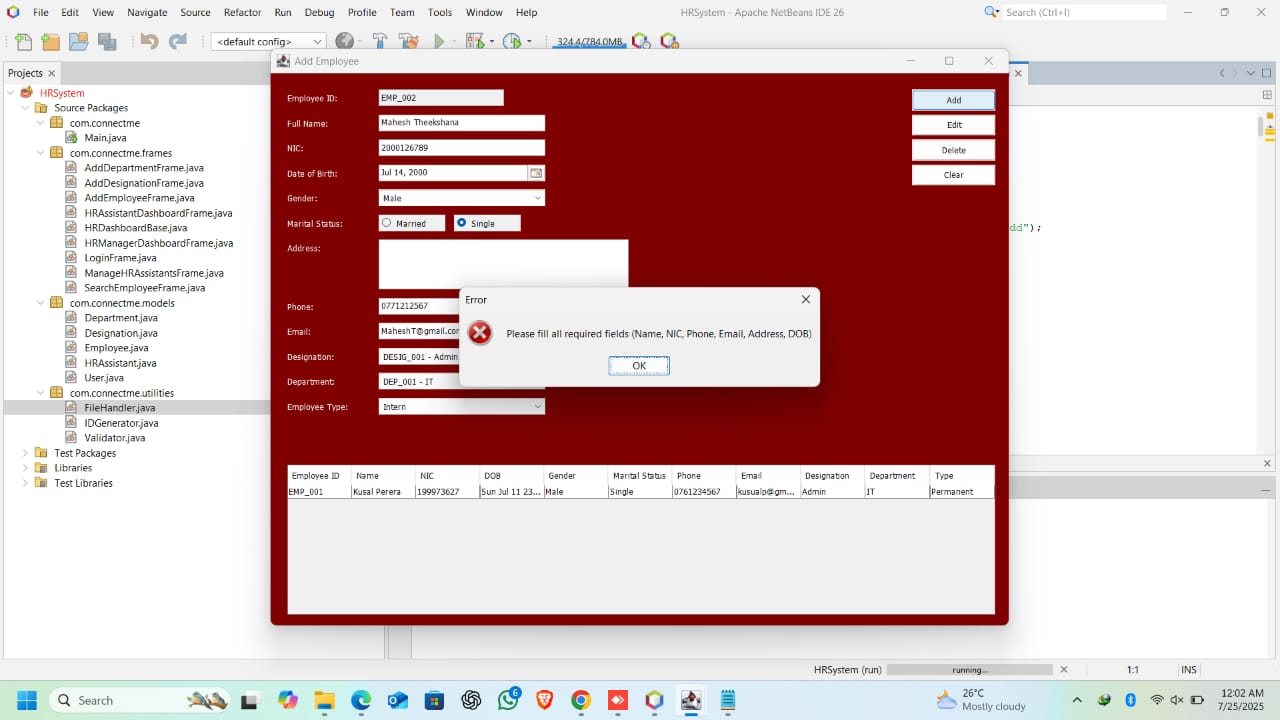
**5. Manager can Add Designations successfully and the data will be stored in “designations” File.**

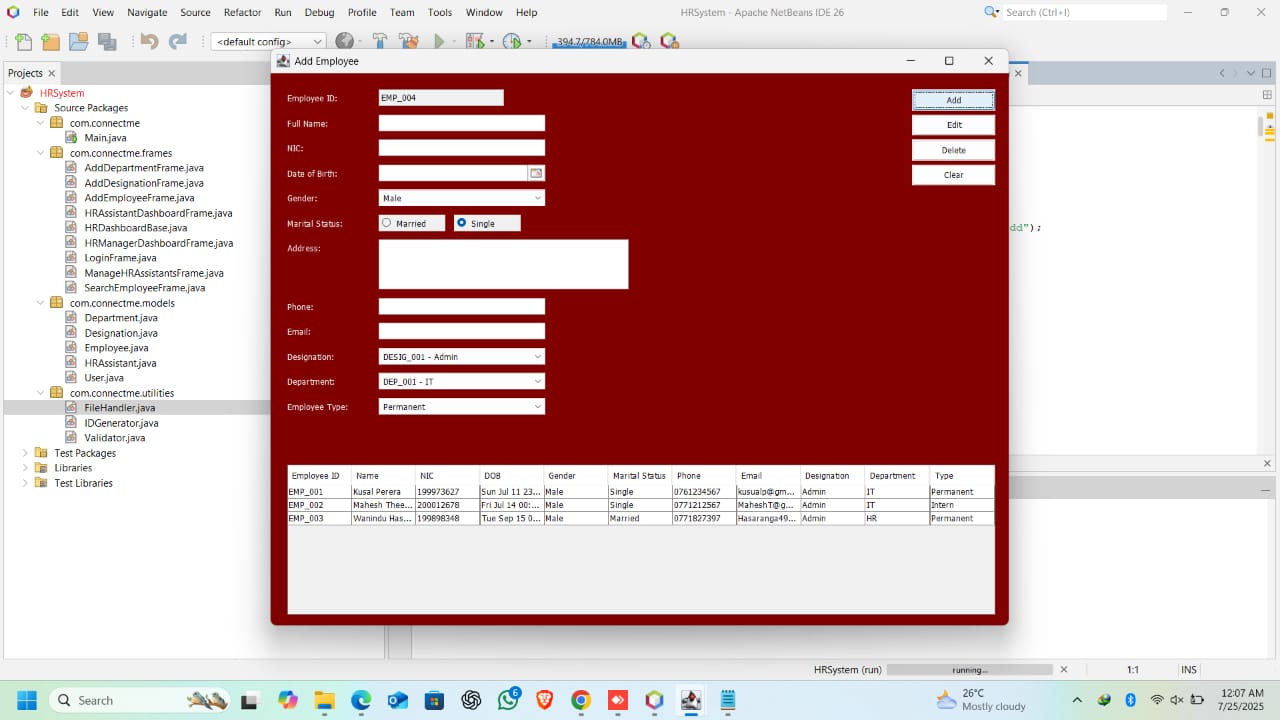
****

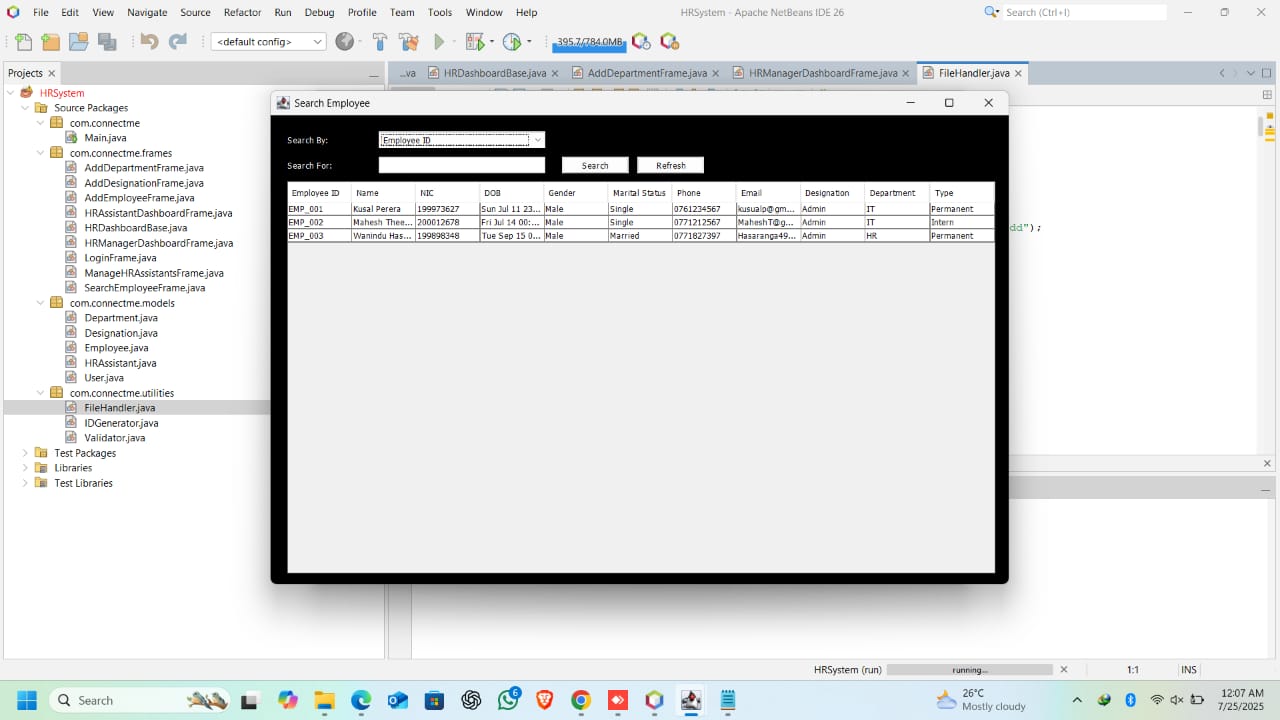
**6. Manager can Add Employees successfully and the data will be stored in “employees” File.**

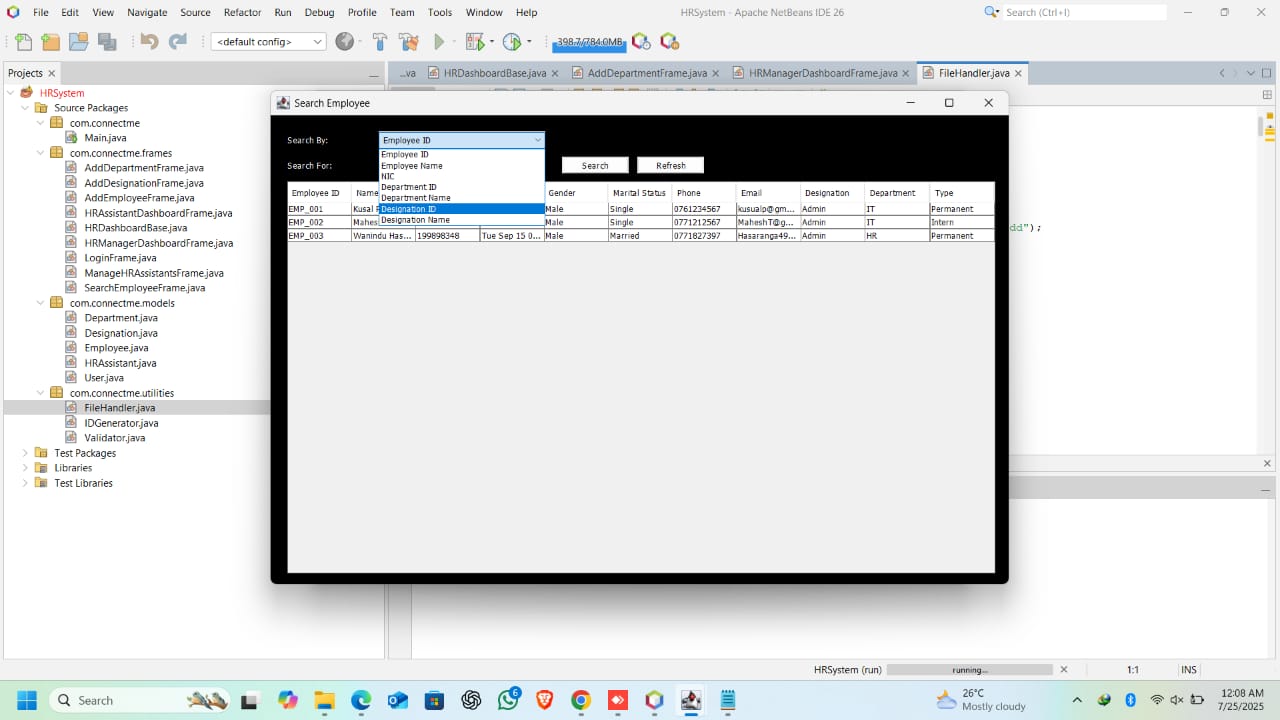
****

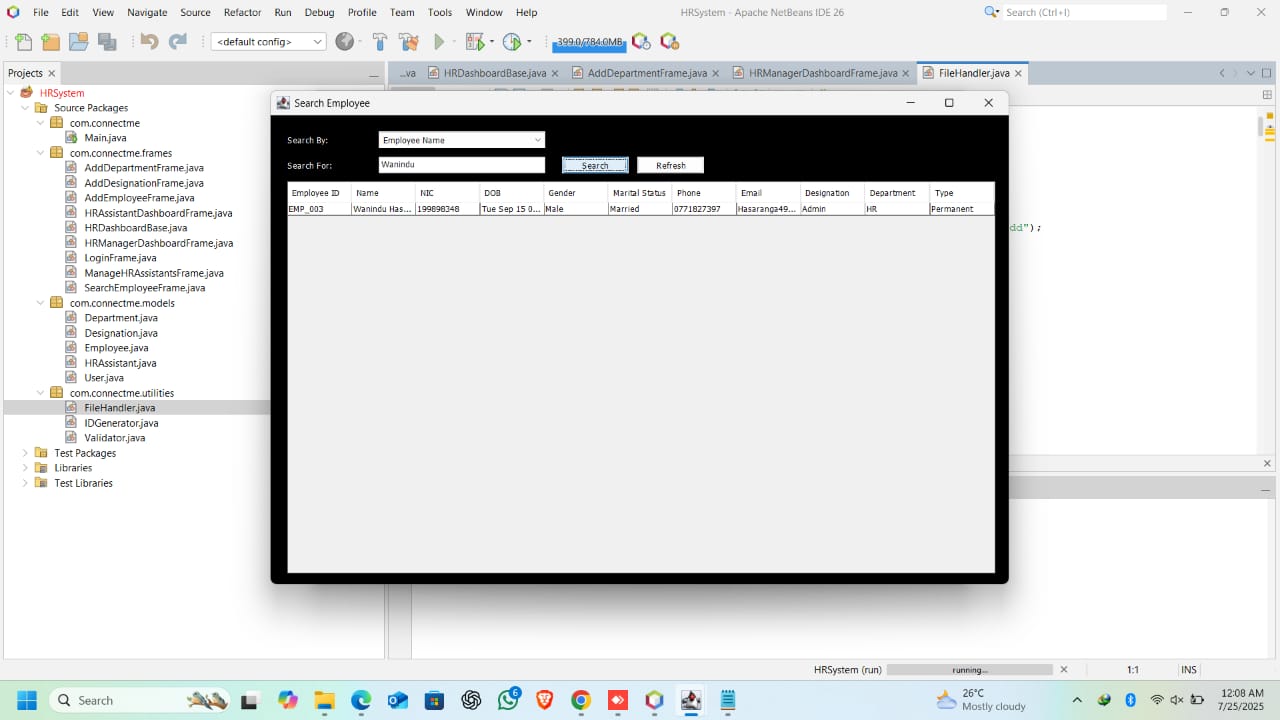
**7. If the phone number of Employee doesn’t have 10 digits, an error message will display.**

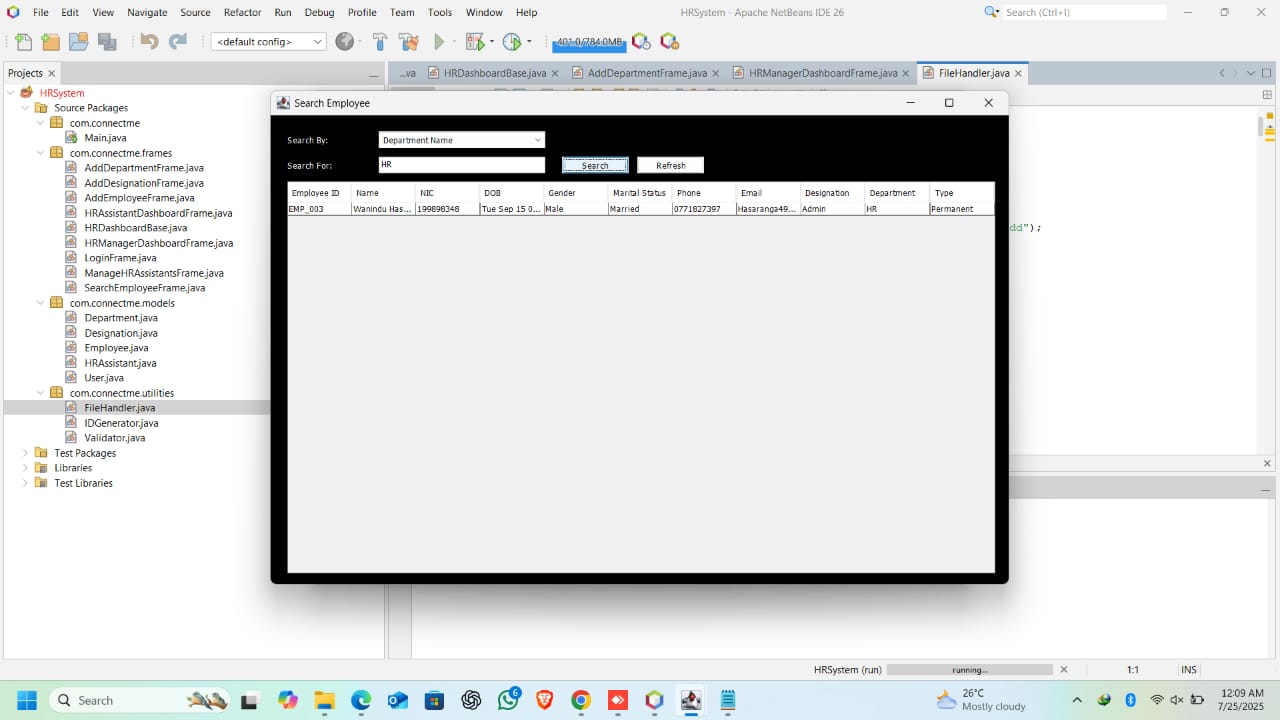
**8. If any field is not filled, an error message will be displayed.**

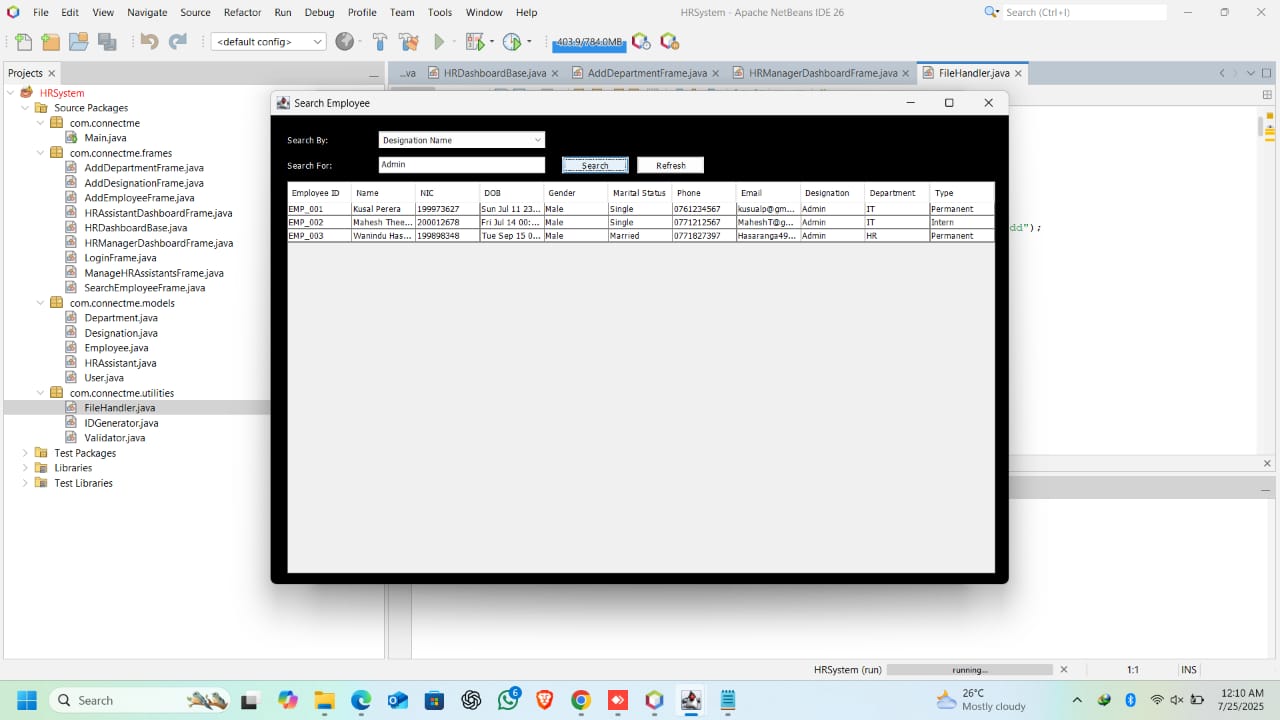
**9. Manager can create any number of employees.**

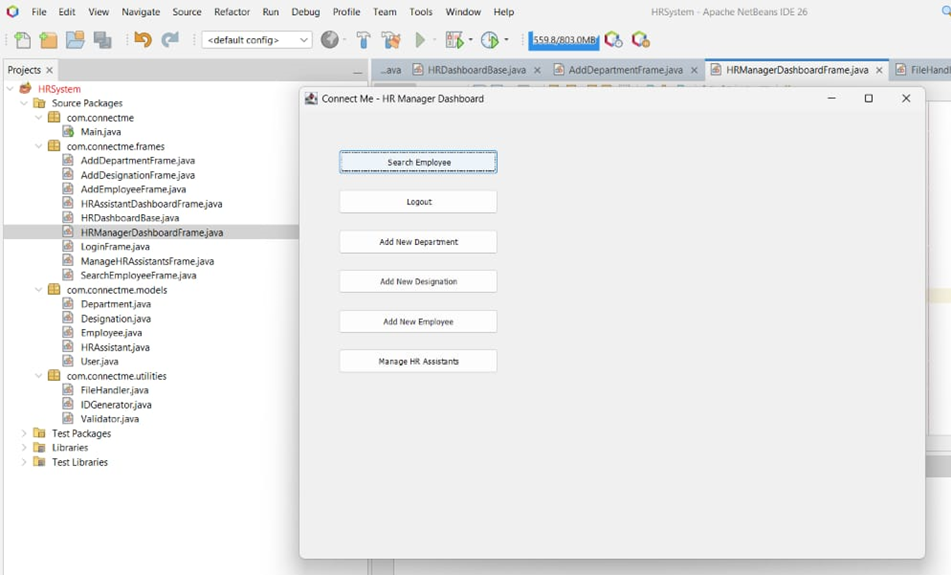
**10. Manager can search their employees by names, department and designation by “Search Employee”.**

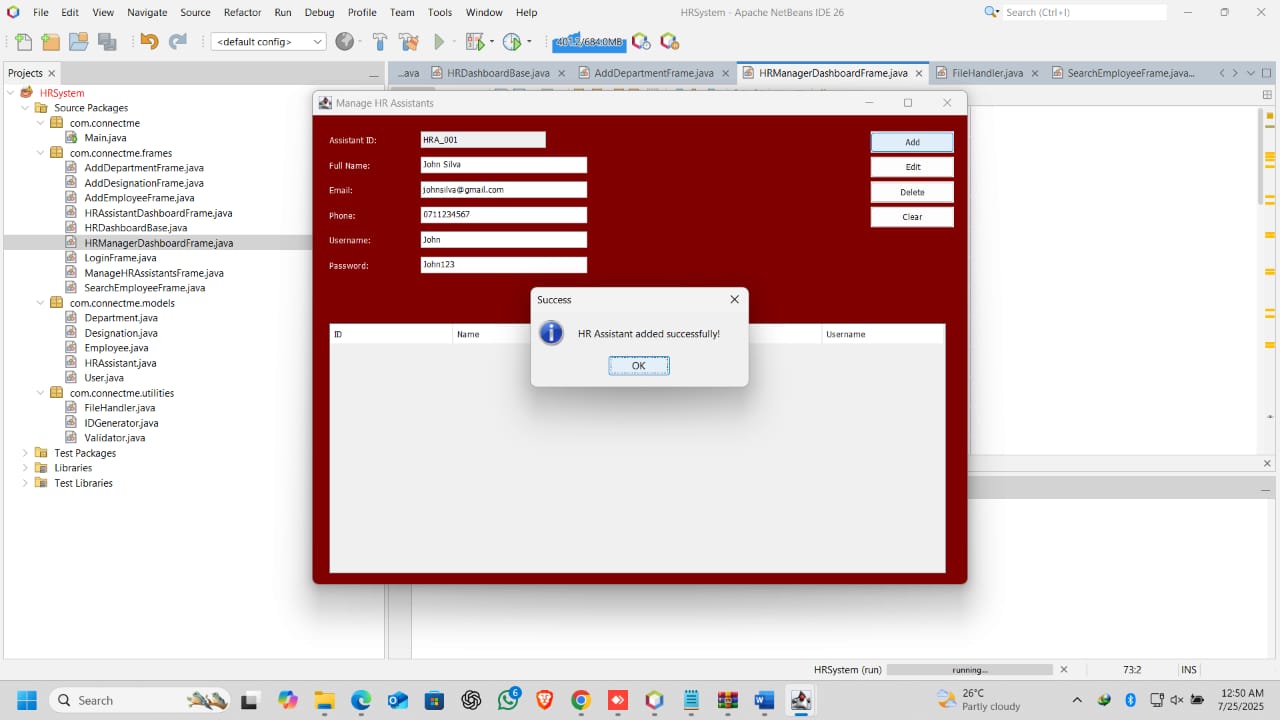
****

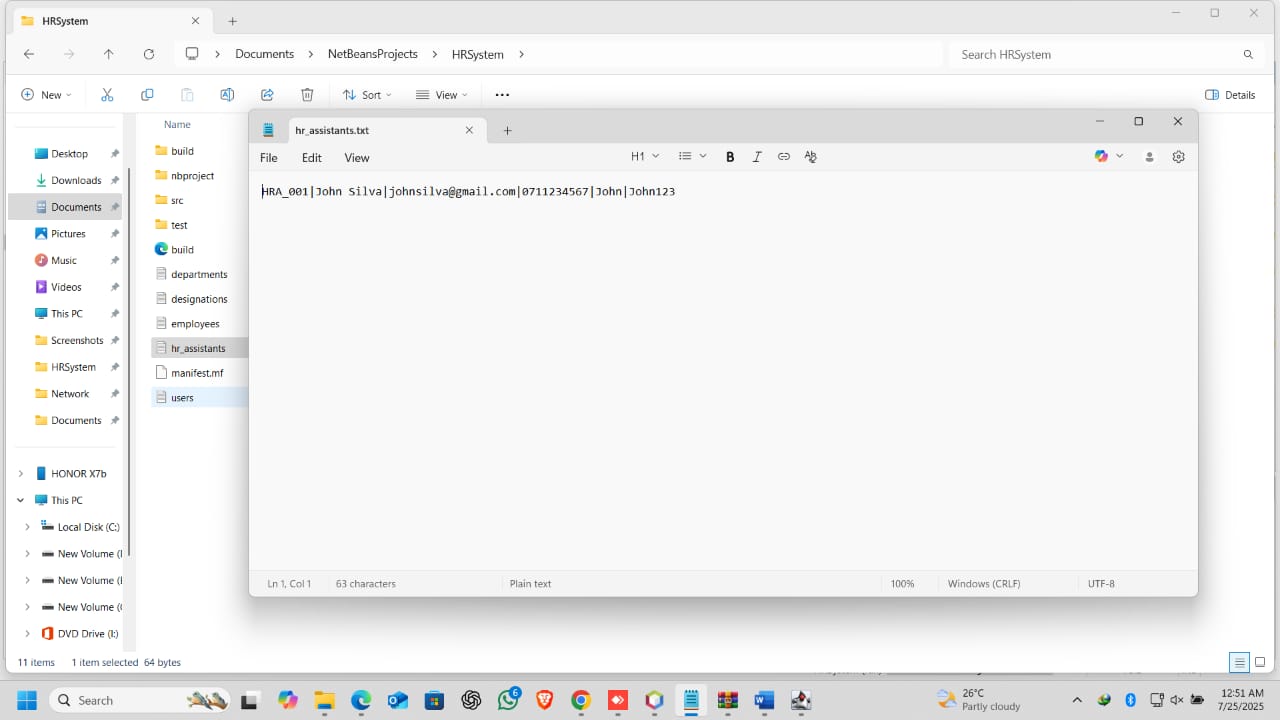
**a) Name**

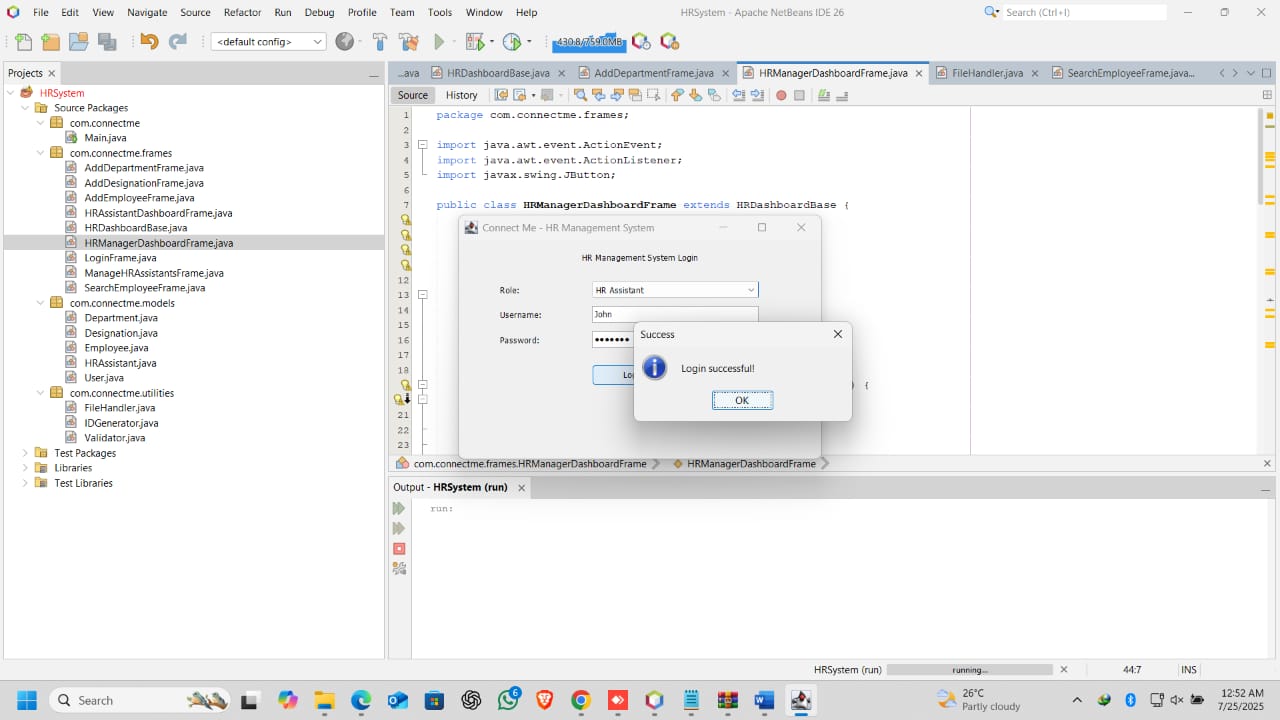
**b) Department**

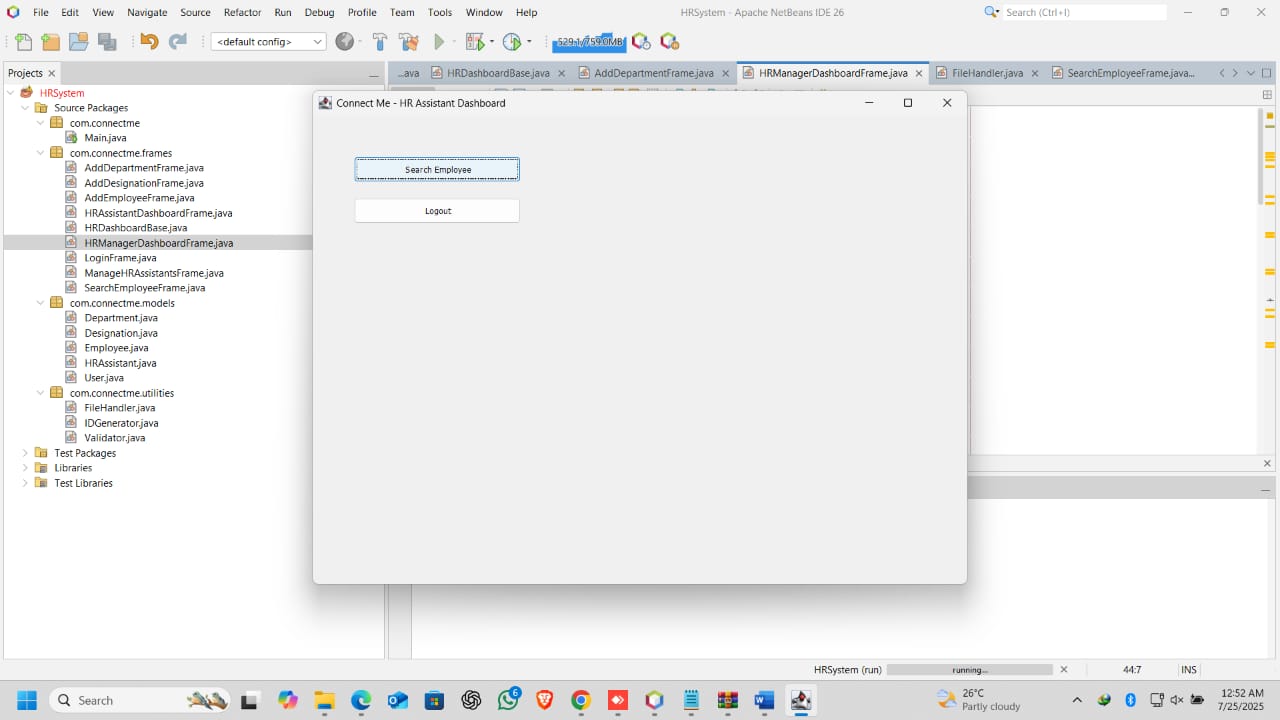
**c) Designation**

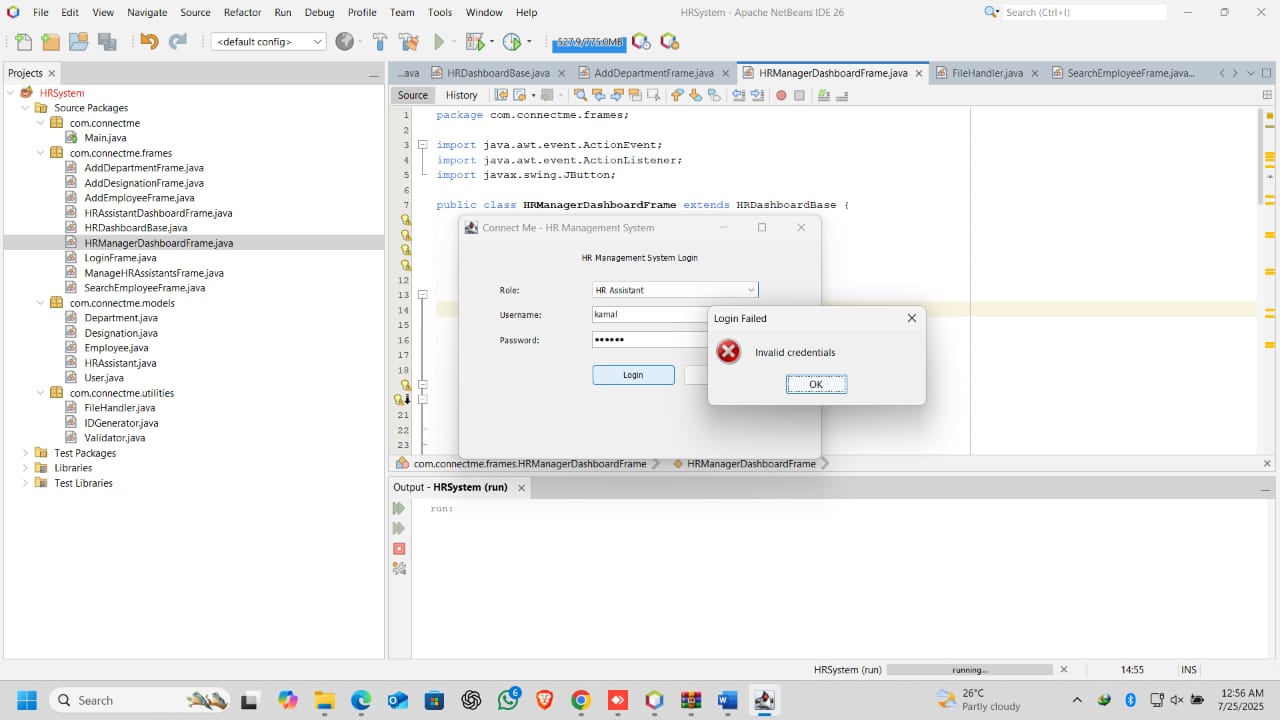
**11. Manager can create the accounts for the HR Assistants by “Manage HR Assistants” and the data will be stored in “hr\_assistants” File.**

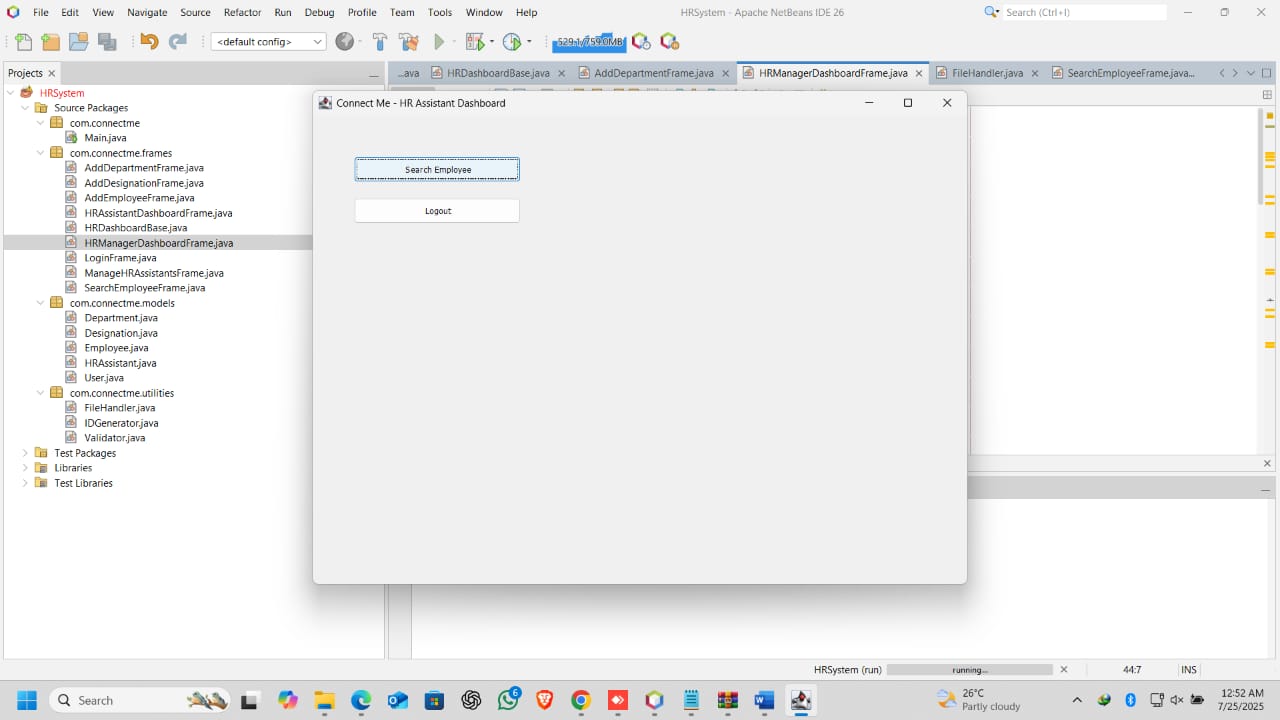
****

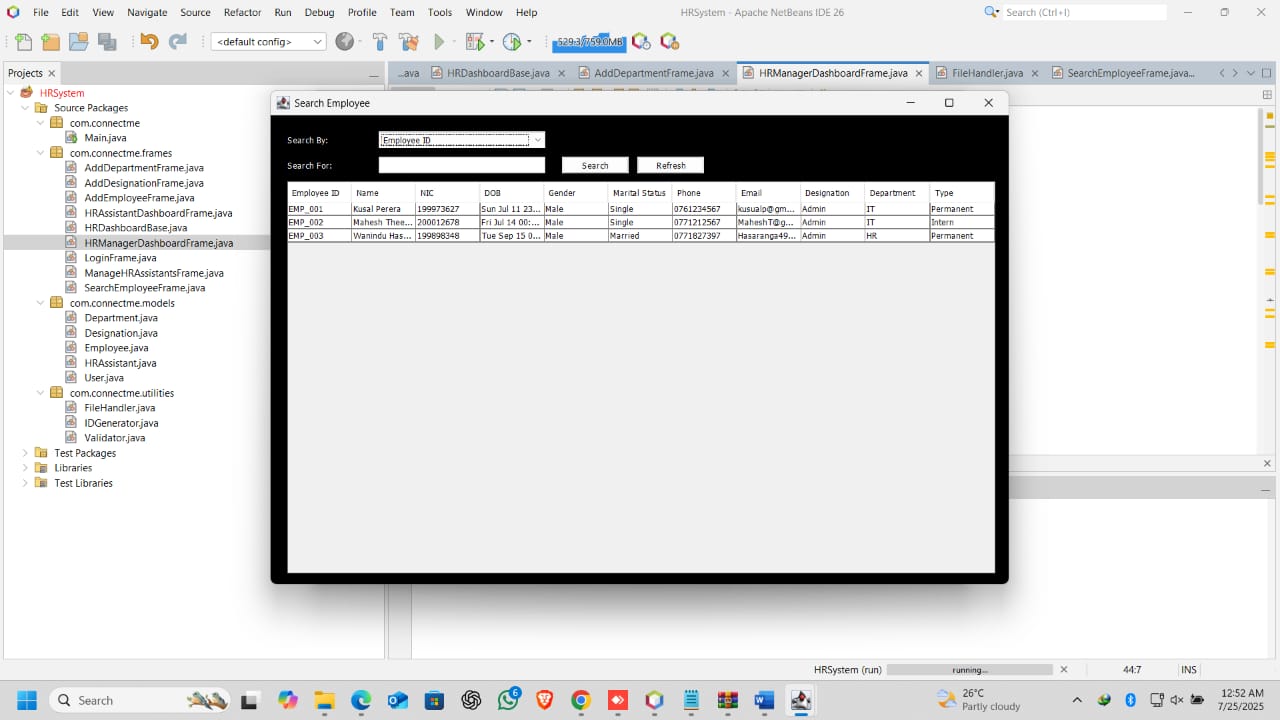
****

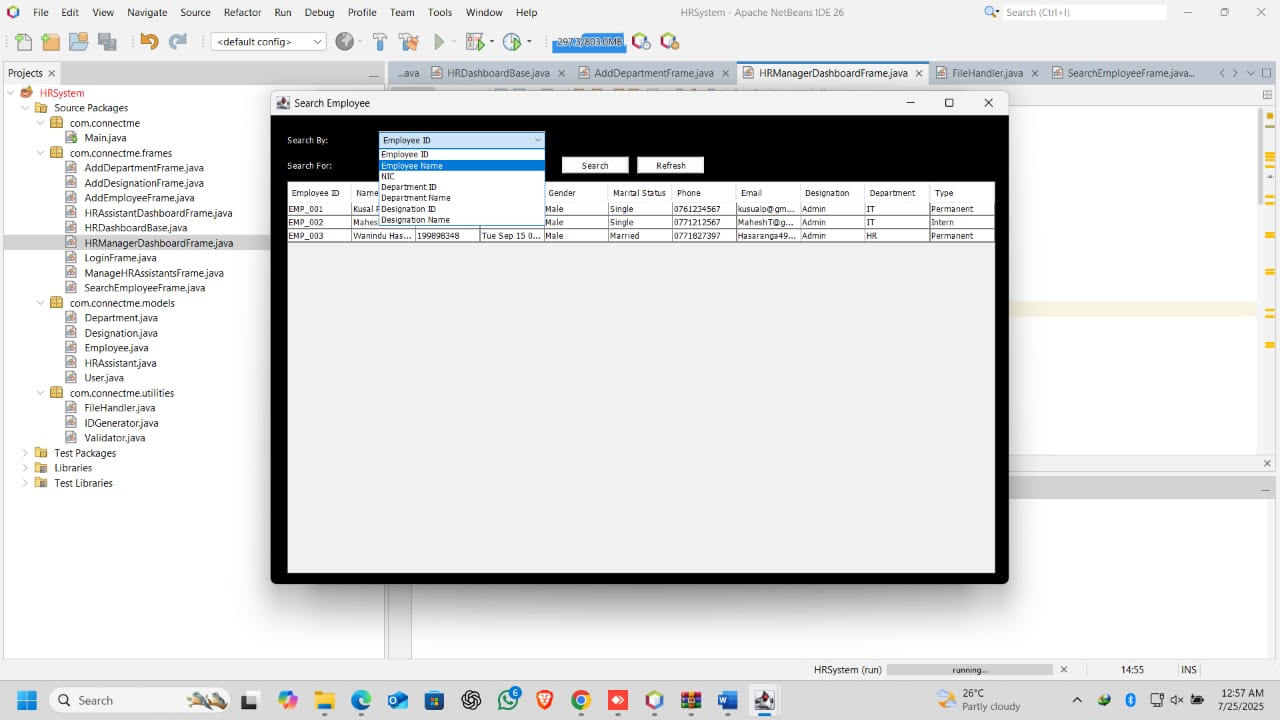
**12. Can login to the created HR Assistant Accounts successfully**

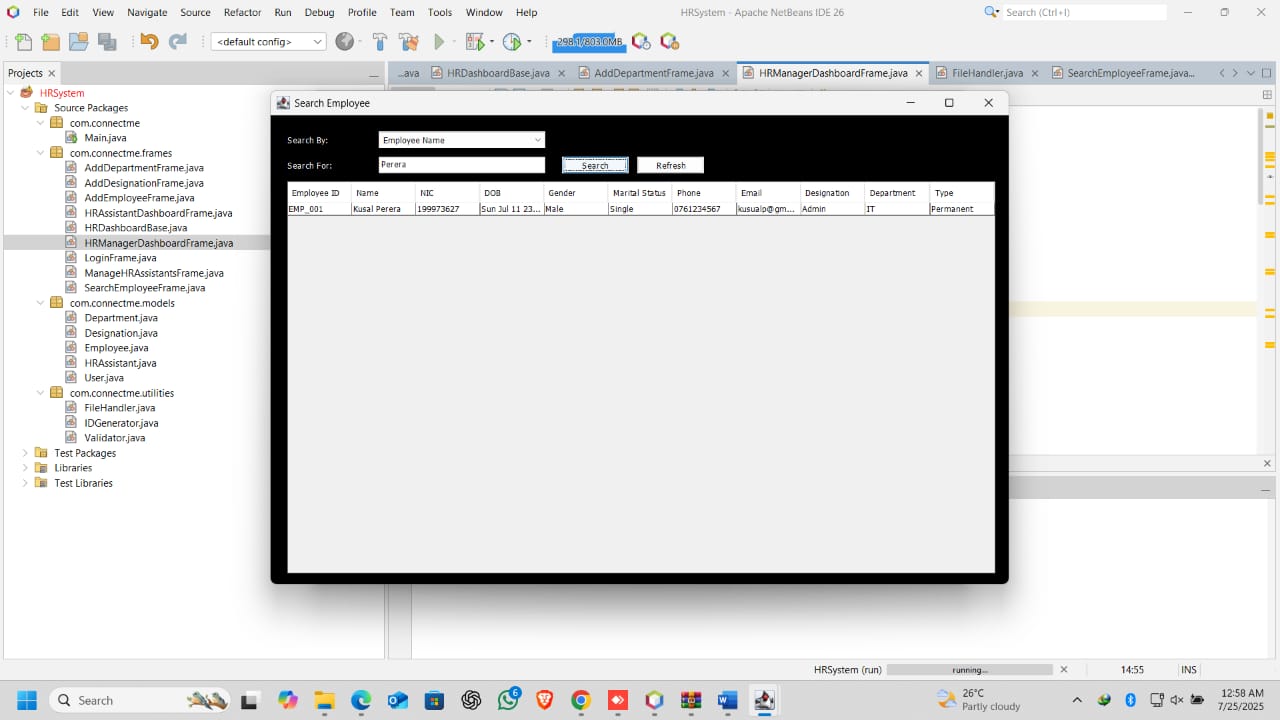
****

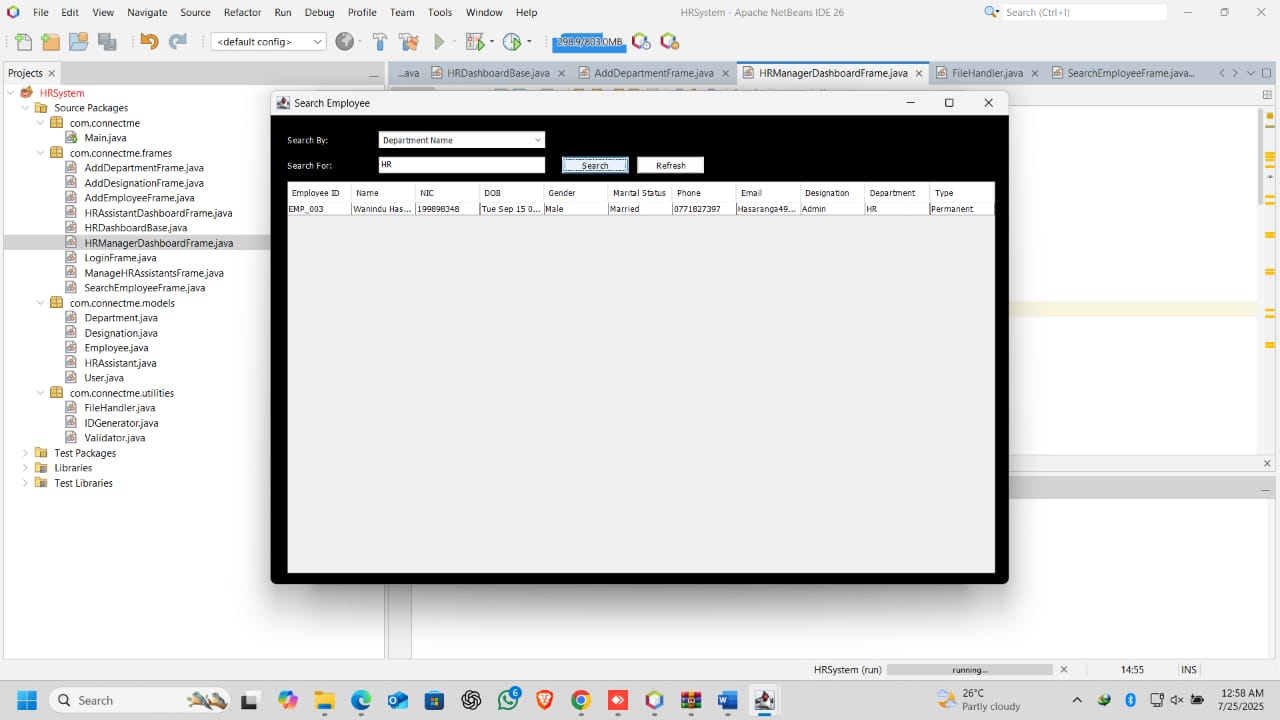
**13. If the Username or Password is incorrect an error message will display.**

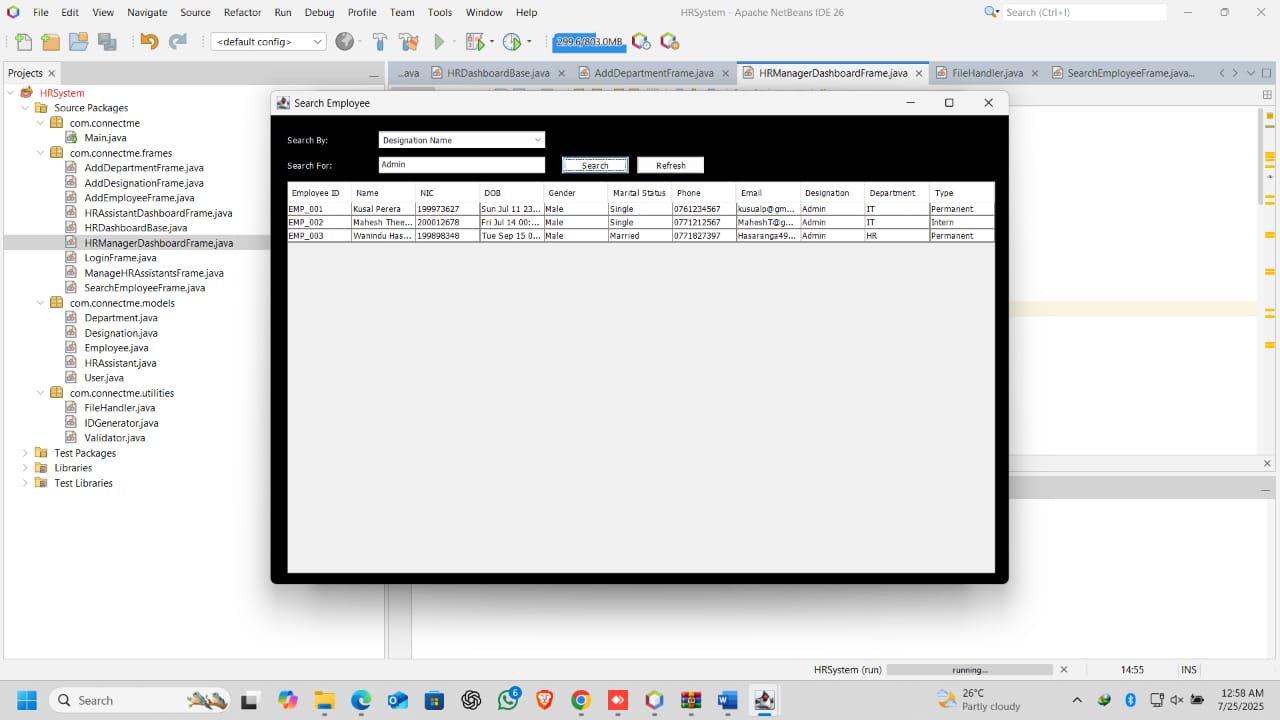
**14. HR Assistants can search Employees**

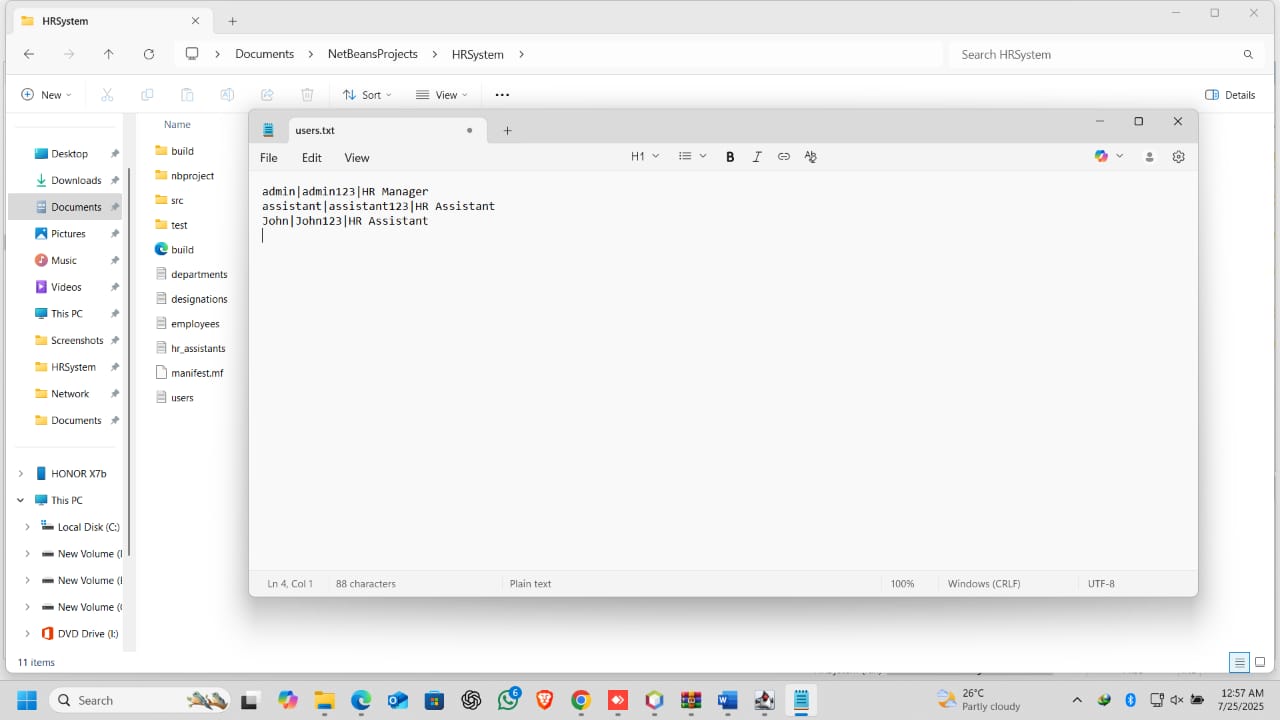
****

****

**a) Name**

**b) Department**

**c) Designation**

**15. All the Logins will be saved in “users” file.**

**How to Exit**

To Exit just click the X button easily to quit the program.

**Troubleshooting**

* 1. If the login fails, check username or password and re-login.
  2. If the file is not found, check if text files are in the relevant folder.
  3. If an error message says to fill all fields, make sure all fields are filled.
  4. If an error message says format is incorrect or phone number is incorrect, make sure the relevant formats are correct (eg: xxxx@gmail.com) and phone numbers have 10 digits (eg: 077xxxxxxx).

**Conclusion**

An example of the application of object-oriented programming in the development of an effective program that caters to future scalable requirements is the use of Java in writing Human Resource Management System (HRMS). By use of OOP essence elements especially encapsulation and inheritance and polymorphism practices the project was able to achieve proper modeling of HR business processes that are comprised of employee management and departmental set-ups leave management and salaries payments systems. The system carries out two basic functions that make complicated HR activities easy and locks down data and inventory in the management of resources. Systematic design approach and program code which is divided into compartments within the development project is important because it forms a stable foundation by which development can occur at a later date and can expand. These HRMS systems will come in handy in modern organizations that are struggling to digitize their operations to attain a better rate of productivity.

**References**

* 1. Oracle (2017). *Lesson: Object-Oriented Programming Concepts (The JavaTM Tutorials > Learning the Java Language)*. [online] Oracle.com. Available at: <https://docs.oracle.com/javase/tutorial/java/concepts/>.
  2. GeeksforGeeks (2023). *Java OOP(Object Oriented Programming) Concepts*. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/java/object-oriented-programming-oops-concept-in-java/>.
  3. Visual Paradigm (2024). *UML Class Diagram Tutorial*. [online] Visual-paradigm.com. Available at: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial>/.
  4. www.programiz.com. (n.d.). *Java File (With Examples)*. [online] Available at: <https://www.programiz.com/java-programming/file>.
  5. Oracle (2019). *Trail: Creating a GUI With JFC/Swing (The JavaTM Tutorials)*. [online] Oracle.com. Available at: <https://docs.oracle.com/javase/tutorial/uiswing/>.
  6. baeldung (2017). *Polymorphism in Java | Baeldung*. [online] www.baeldung.com. Available at: <https://www.baeldung.com/java-polymorphism>.
  7. W3Schools (2019). *Java Inheritance (Subclass and Superclass)*. [online] W3schools.com. Available at: <https://www.w3schools.com/java/java_inheritance.asp>.