Group 13

Subject: CT24-3-3-DCOMS

A HUMAN RESOURCE MANAGEMENT (HRM) SYSTEM FOR PENTAFOUR

HAND OUT DATE: 19TH DECEMBER 2024

HAND IN DATE (GROUP ASSIGNMENT) : 1ST MARCH 2025

Name	TP Number	Intake
DARSHRNNEY A/P PUSHPANATHAN	TP072253	APD3F2411CS
HABEBA MOHAMED ABDELSALAM MOHAMED HASSAN	TP067863	APD3F2409CS
ANGELITA DEBORAH A/P CHARLES TERRENCE LOURDES	TP062131	APD3F2411SE
LUBAINA ABDULLAH KHAN	TP072033	APU3F2411CS

TABLE OF CONTENTS

1.0 GANTT CHART

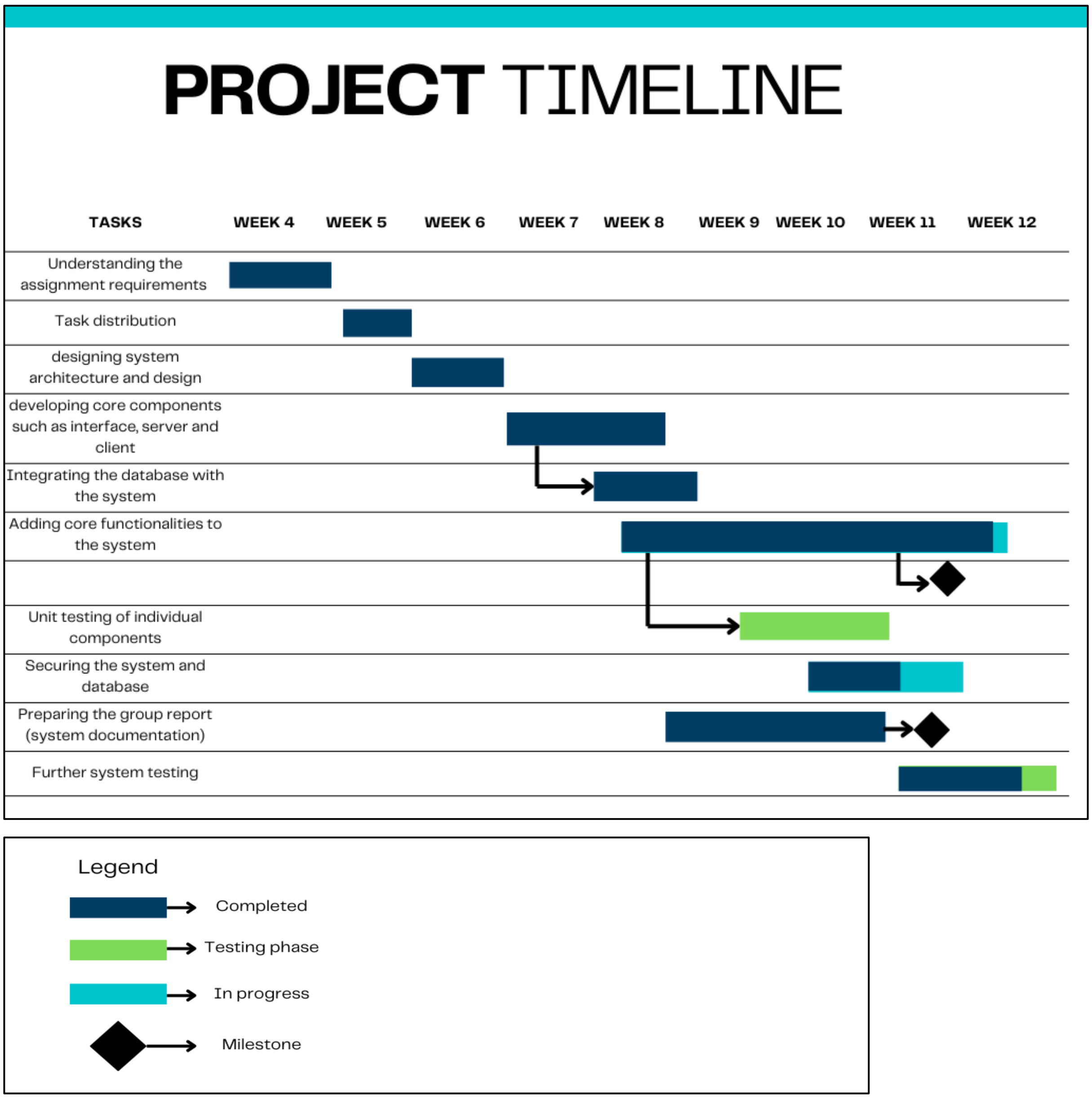


Figure 1.0: Gannt Chart of the entire project

2.0 ABSTRACT

PENTAFOUR currently uses manual paper records and spreadsheets to manage employee leaves, resulting in errors, delays, and inefficiencies. This report outlines a distributed HRM system using Remote Method Invocation (RMI) technology to create a client-server application. The system enables employees to update profiles, check leave balances, apply for leave, and view application status. HR staff can register employees and generate reports. The system uses a JavaDB database for centralized data management.

The report also examines multi-threading, serialization, and object-oriented programming in distributed systems, and provides a fault-tolerant application evaluation based on usability, maintainability, and heterogeneity. The report includes use case modelling, explains RMI protocols, provides a project timeline, and justifies the usefulness of the distributed system compared to a centralized system in the context of PENTAFOUR's requirements.

3.0 INTRODUCTION

PENTAFOUR, a growing startup, seeks to implement a computerized, modern Human Resource Management system to ease the employee leave management processes and enhance application accuracy. This paper provides a distributed HRM solution tailored to PENTAFOUR's needs, using Remote Method Invocation technology to create a client-server application enabling employees to manage profiles, check leave balances, and track applications while allowing HR staff to register employees and generate reports. The implementation addresses multi-threading, serialization, and object-oriented programming in distributed systems with future recommendations for cloud computing and virtualization enhancements.

4.0 PROBLEM BACKGROUND

PENTAFOUR's paper-based and spreadsheet-based leave management system has been increasingly troublesome as the business grows. The manual process creates numerous challenges affecting overall efficiency and satisfaction.

Manual record-keeping is highly susceptible to human errors, leading to inconsistencies in employee leave data (Epen et al., 2022). These errors lead to incorrect leave balances, mismanaged applications, payroll issues, and potential compliance violations with legislative leave regulations (Sonar & Pandey, 2023).

The system is extremely labour-intensive, whereby HR staff have to process applications manually, update files, and individually notify employees about application outcomes. Such a personalized approach—either face-to-face or through customized emails—is time-consuming, reducing HR productivity and resulting in processing delays that frustrate employees, who may need to plan their leave well in advance for family or personal commitments (Abuhantash, 2023).

Report generation is also tedious, as HR staff must gather and compile data from multiple spreadsheets and paper documents, which is not just time-consuming but also prone to errors (Abuhantash, 2023). Without a centralized database, generating accurate reports for decision-making, auditing, and compliance requirements becomes challenging.

Moreover, the system lacks scalability, making increasingly difficult to manage as the company and employee count grows (Porkodi & Raman, 2024). Physical records reliance results in storage, retrieval, and maintenance difficulties that bottleneck HR operations.

Given these limitations, PENTAFOUR feels the need for a modern, computerized Human Resource Management (HRM) that addresses the shortcomings of the manual system. The proposed distributed system aims to improve efficiency, accuracy, and transparency in leave management, ultimately improving employee satisfaction and supporting company growth.

5.0 REQUIREMENTS

To overcome the issues that PENTAFOUR is encountering and ensure that the suggested HRM system has a successful implementation, the following requirements need to be met:

- Employee registration: The system should provide a user-friendly interface for HR staff to register employee data, including essential information such as First Name, Last Name, and IC/Passport. This functionality will ensure that all employee records are accurately captured and stored in the centralized database, eliminating the need for manual data entry and reducing the risk of errors.

- Employee Self-Service: A key feature of the HRM system should be the implementation of a self-service portal that allows the employee to view and manage their personal data, leave balances, and leave applications. Employees should be allowed to update their profile and family data, check leave balances, apply for leave, and see the status of their leave applications and leave history. The self-service function will help empower employees to manage their leave effectively, consequently reducing the burden on HR staff and increasing total efficiency.

- Centralized Database: The system should be able to store and manage all employee information, including personal data, leave balances, and leave history, in a centralized database. All data will be considered consistent, accessible, and secure in a centralized database, thus eliminating the need for human record-keeping and, with it, the chances of data loss or inconsistencies. The design of the database must be done considering the scalability requirements of the organization so that in the future, an increase in the number of staff and complexity of leave management can be easily accommodated.

- Report Generation: The HRM system should have a robust reporting module capable of generating yearly reports with in-depth information on employee leave history for HR staff. The reporting should allow custom reports according to various parameters such as employee name, leave type, and date. These reports will be good in explaining the leave trends, identifying patterns, and backing the decision-making process regarding leave management and workforce planning.

- Secure Communication: Owing to the sensitive nature of employee information and leave data, the Human Resource Management (HRM) system must ensure secure communication between employees, HR staff, and the system itself. Thus, other security measures appropriate for ensuring data privacy and preventing unauthorized access should be put in place: encryption, authentication, and access controls. Secure communication would help derive trust from employees and ensure adherence to data protection regulations.

- User-Friendly Interface: The HRM system should have a user-friendly interface that is easy to navigate by both the employees and the HR staff. The interface should be developed around the end-user perspective: giving concise instructions to guide users through the various functionalities of the system. User-friendly interfaces would minimize training, reduce operational errors, and increase adoption of the system throughout the organization.

- Workflow Automation: The system should handle workflows related to leave management in an automated manner, including workflow processes such as submitting leave applications, approvals, and rejections. Workflow automation will facilitate ease of use in leave management, eliminate manual intervention, and ensure timely processing of leave requests.

- Integration Abilities: The design of the HRM system must keep in mind the fact that this will be an integrated solution with other future or existing HR systems. Data sharing can help improve efficiency for the entire HR operations by eliminating data silos while providing a complete view on employee information.

By meeting these requirements, PENTAFOUR can ensure that the proposed HRM system would effectively address and provide a strong, efficient, and user-friendly solution for employee leave records management as all address limitations of the current manual system.

6.0 RESEARCH AND EVALUATION

6.1 RMI (Remote Method Invocation)

Remote Method Invocation (RMI) is a distributed system when the systems is separated and uses a remote machine or the same machine that allows an API to invoke a method from this object to another, in a separate address space. The RMI set up can be separated into two areas, the Client-side and Server-side. For example, the Automated Teller Machine (ATM) uses the RMI method to fetch objects written in the server side and display them through the client's side. All processes are done in real time and instantaneously. There should be no delay when the clients receive their information from the server. When RMI occurs , there is a need for serializing and desterilizing the transmission to maintain security of the system. Below is the process of how RMI works (M. Mihailescu, 2015).

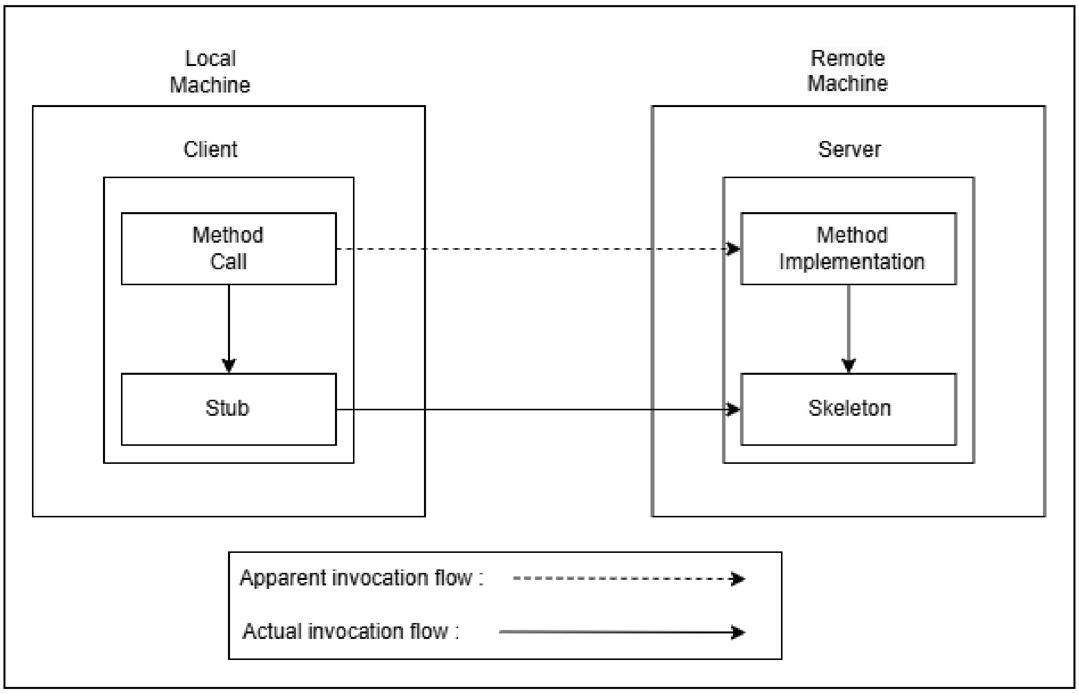
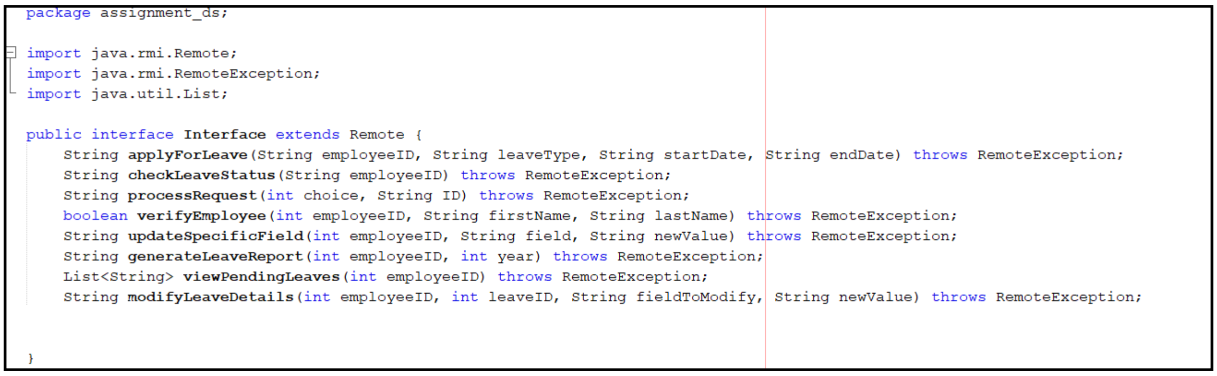


Figure 1.1: RMI Diagram

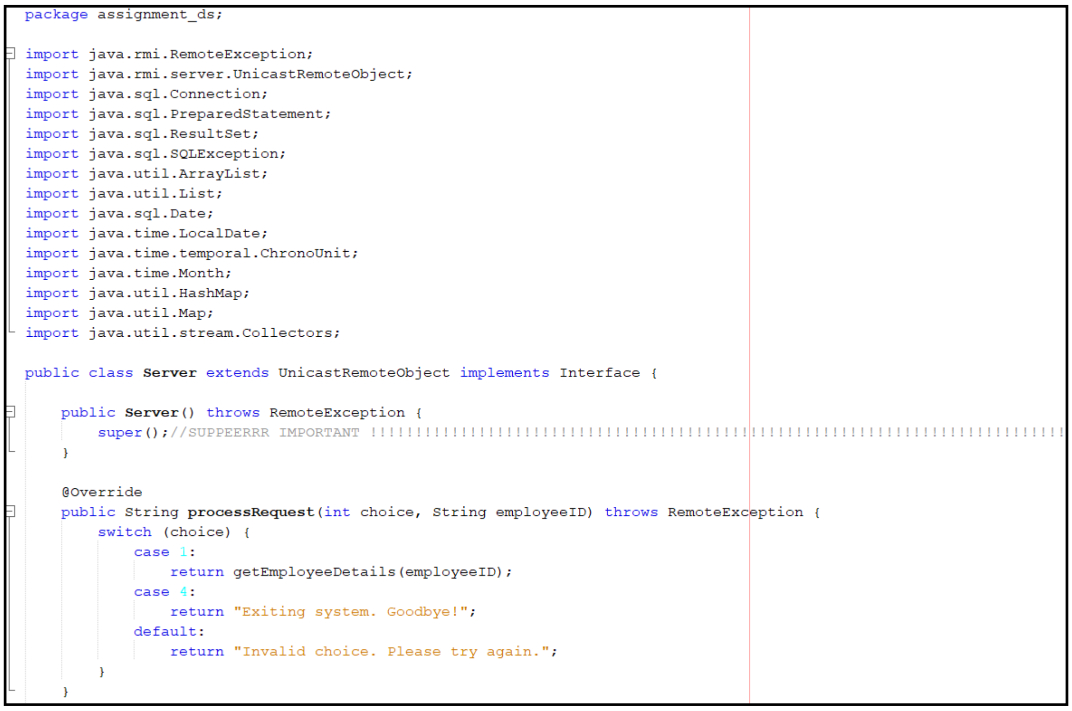
In our system, we first define an interface that imports the java.rmi package into the file. This interface extends the Remote interface, allowing its methods to be accessible remotely. The client class, which incorporates serialisation, extends this interface. Furthermore, each method in the interface is intended to throw a RemoteException, ensuring proper handling of remote communication problems. This organised technique allows for easy remote method invocation while retaining dependability in distributed system communication (M. Mihailescu, 2015).



```java
package assignment_ds;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface Interface extends Remote {
    String applyForLeave(String employeeID, String leaveType, String startDate, String endDate) throws RemoteException;
    String checkLeaveStatus(String employeeID) throws RemoteException;
    String processRequest(int choice, String ID) throws RemoteException;
    boolean verifyEmployee(int employeeID, String firstName, String lastName) throws RemoteException;
    String updateSpecificField(int employeeID, String field, String newValue) throws RemoteException;
    String generateLeaveReport(int employeeID, int year) throws RemoteException;
    List<String> viewPendingLeaves(int employeeID) throws RemoteException;
    String modifyLeaveDetails(int employeeID, int leaveID, String fieldToModify, String newValue) throws RemoteException;


}
```

Figure 1.2: Interface Class

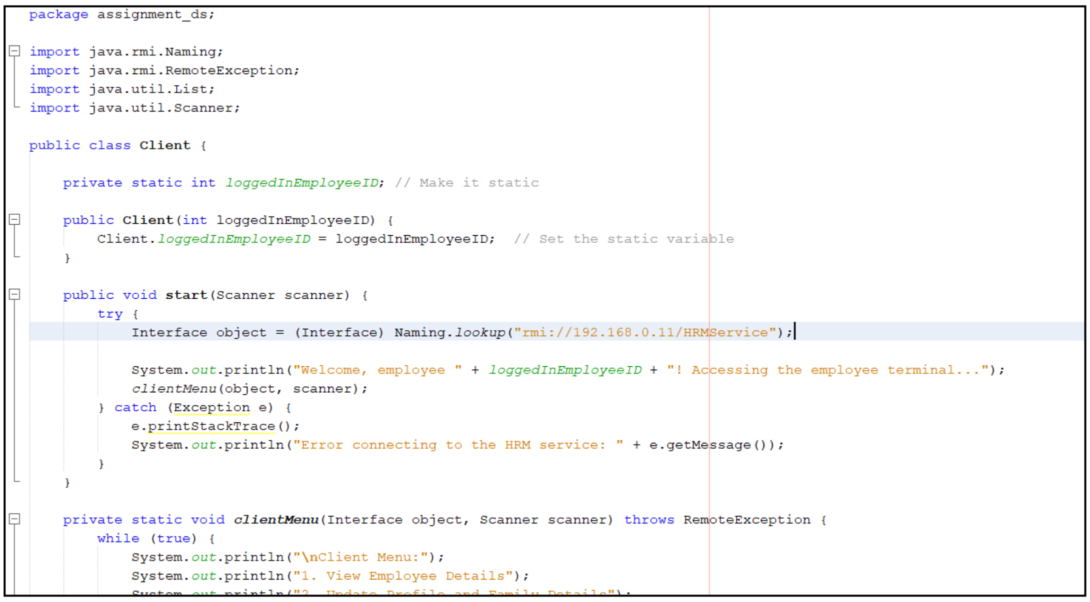
Next the server class is created to implement its interfaces. This class would extend UnicastRemoteObject method, which enables point-to-point communication over Transmission Control Protocol (TCP) layer. The constructer then throws the RemoteException, to ensure proper handling of remote method calls between the client and server. Various methods are created in the class, such as getting employee details or applying leave to return values to the employees (M. Mihailescu, 2015).



```java
package assignment_ds;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.sql.Date;
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.time.Month;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

public class Server extends UnicastRemoteObject implements Interface {

    public Server() throws RemoteException {
        super();//SUPPEERRR IMPORTANT !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    }

    @Override
    public String processRequest(int choice, String employeeID) throws RemoteException {
        switch (choice) {
            case 1:
                return getEmployeeDetails(employeeID);
            case 4:
                return "Exiting system. Goodbye!";
            default:
                return "Invalid choice. Please try again.";
        }
    }
```

Figure 1.3: Implementing Methods & Interfaces

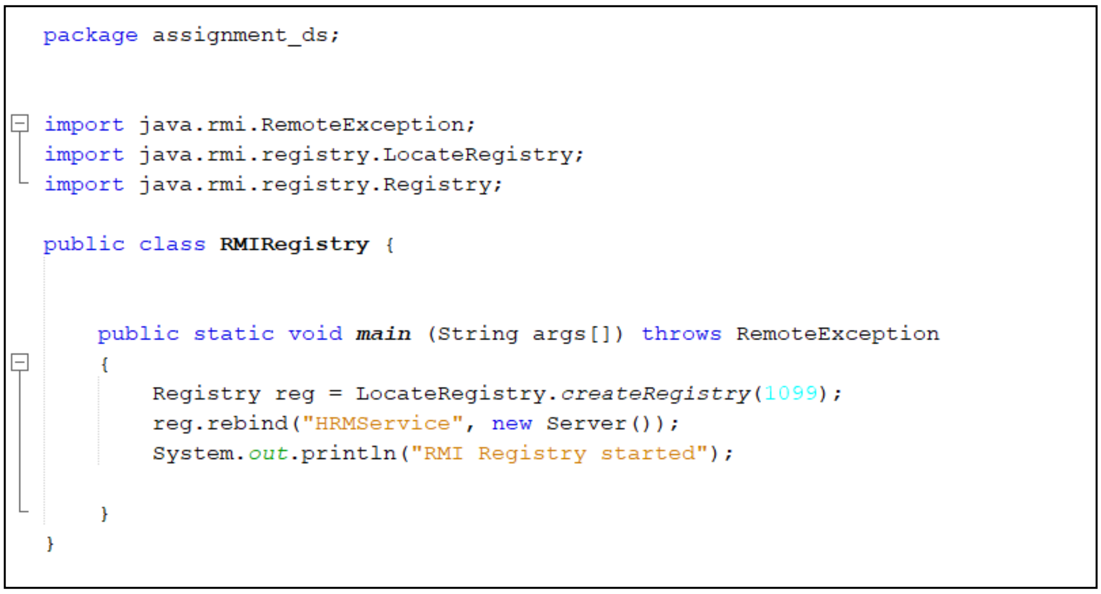
Furthermore, to instantiate the client class we must register it with the registered service URL by using Naming.rebind() method. This URL form is used, allowing customers to look up specific object (M. Mihailescu, 2015).



```java
package assignment_ds;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.util.List;
import java.util.Scanner;

public class Client {

    private static int loggedInEmployeeID; // Make it static

    public Client(int loggedInEmployeeID) {
        Client.loggedInEmployeeID = loggedInEmployeeID;  // Set the static variable
    }

    public void start(Scanner scanner) {
        try {
            Interface object = (Interface) Naming.lookup("rmi://192.168.0.11/HRMService");

            System.out.println("Welcome, employee " + loggedInEmployeeID + "! Accessing the employee terminal...");
            clientMenu(object, scanner);
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Error connecting to the HRM service: " + e.getMessage());
        }
    }

    private static void clientMenu(Interface object, Scanner scanner) throws RemoteException {
        while (true) {
            System.out.println("\nClient Menu:");
            System.out.println("1. View Employee Details");
```

Figure 1.4: Naming.lookup Implementation

Following this, the registry class is created to allow server to register its remote objects and client to retrieve it. It legally binds them together making it accessible over the network. We are using port 1099 , which a default port in RMI to let clients look up objects. The process is complete as, all the classes come together and displays the output of the object selected (M. Mihailescu, 2015).



```
package assignment_ds;


import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class RMIRegistry {


    public static void main (String args[]) throws RemoteException
    {
        Registry reg = LocateRegistry.createRegistry(1099);
        reg.rebind("HRMService", new Server());
        System.out.println("RMI Registry started");

    }
}
```

Figure 1.5: RMI Registry

6.2 DCOM (Distributed Component Object Model)

DCOM is an extension for Component Object Model (COM) that enables software from Microsoft to communicate with various types of Local Area Network (LAN) in the area or even WAN for larger scales. For multi connection the use of internet can be used to distribute the COM-based applications.

Meanwhile, COM is used to be the standardization of the whole system , DCOM is used to extend this simple function using network protocols, allowing distributed applications to easily be connected to one another. This is based on the DCE/RPC protocols, in converting memory data structure into network packets. One of DCOM unique feature is that it can support and transport protocols by marshalling while optimizing various single remote procedure call (RPC). Additionally, DOM has an automated garbage collection like Java Virtual Machine (JVM) does to clean and manage a safer memory storage system. However, since its ties to a window platform it doesn't have much recognition as other distributed systems (W. Steve , 2019).

The main reason why RMI is better than DCOM is because of the limiting factor of the Microsoft environment that limits it compatibility to other system. While DCOM contains extensive configuration and setups , it also contains security issues that makes it unsecure to use this for important organizations regarding of sensitive information (W. Steve , 2019).

6.3 SOAP (Simple Object Access Protocol)

SOAP is a lightweight, XML based protocol that is used for exchanging structured information in decentralized, distributed computer systems that was designed in 1998. SOAP is a formal protocol with strict rules for the message structure and it follows the XML format only. SOAP is platform independent and is quite flexible as it allows developers to develop SOAP APIs (application programming interfaces) in different languages while adding different features and functionalities.

It is built on top of HTTP (Hypertext Transfer Protocol) and supports a wide range of other protocols like SMTP, TCP and JMS which makes it very transport neutral and since HTTP is installed on most systems, SOAP can handle communications without requiring additional configuration. It also includes built in standards such as WS-security which ensures message integrity and secure transmissions. SOAP communications works like this, first the client sends an XML request to the SOAP server and the server responds with the details as a service invocation in XML format to the client (*Basics of SOAP - Simple Object Access Protocol*, 2019).

The SOAP message structure contains of four main parts; Envelope, header, body and fault. The envelope specifies that the XML message is a SOAP message and defines the start and the end of the message. The header is optional and contains additional information about the message such as the authentication credentials. The body contains the actual data (the response or the request) and finally the fault which is optional as well handles any faults that are generated by sending error messages (*What Is SOAP API: Formats, Protocols, and Architecture*, 2019).

While SOAP has many advantages such as being lightweight, transport neutral and reliable messaging, it is quite slow and inflexible compared to RMI and is more complex to implement as it uses WSDL (Web Services Description Language) for defining service contracts.

6.4 CORBA (Common Object Request Broker Architecture)

CORBA is a client-server development model and a middleware technology that was developed in 1992 by the Object Management Group (OMG) which allows applications written in multiple languages to communicate with each other despite their differences. Basically, CORBA creates a unified structures that enables applications with different technologies to communicate.

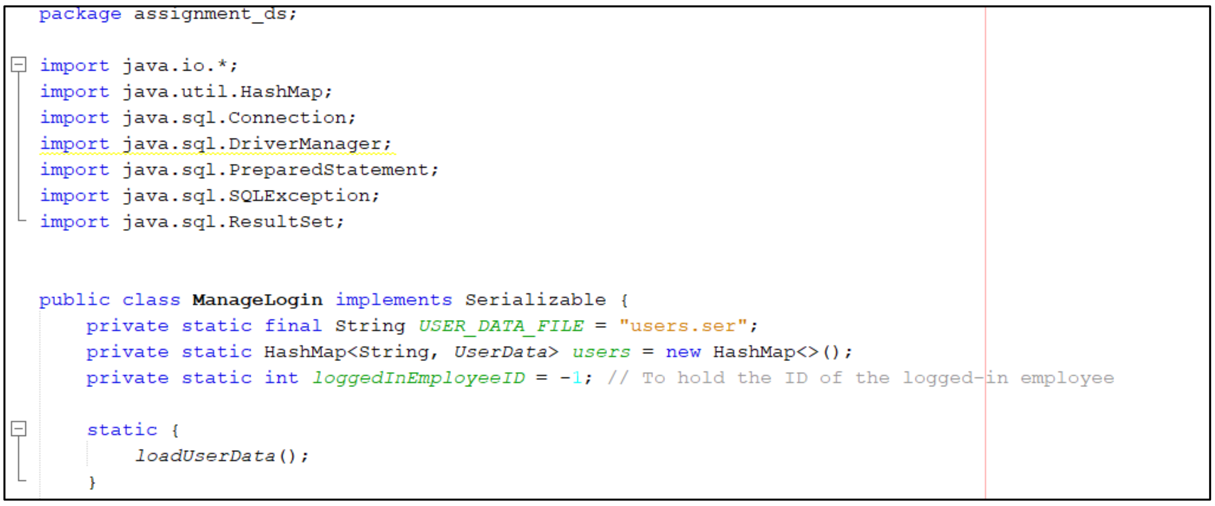
The key component of the CORBA framework is the ORB (Object Request Broker), The ORB acts as a mediator, it implements the object structure also known as the object bus and it handles communication between distributed objects (client and server) transparently regardless of their location or platform. This allows the client to invoke an object (local or remote) and this request is intercepted by the ORB, the ORB serializes(marshals) the method call into a standard format and sends the request to the server on the ORB side which unmarshals the request and serializes the response and returns the response to the client side and the client unmarshals the response and returns the result to the calling application. The ORB is able to do this using the IDL.

The advantages of using CORBA is that it allows reusability of software components, allows interoperability between different platforms and languages and enables clients to invoke objects locally or remotely. However, RMI is simpler to implement since it’s Java native and is faster for java-java communication.

7.0 THE ROLE OF MULTI-THREADING, SERIALIZATION, AND OBJECT-ORIENTED PROGRAMMING (OOP) IN SOLVING PROBLEMS IN DISTRIBUTED SYSTEMS.

7.1 ROLE OF SERIALISATION

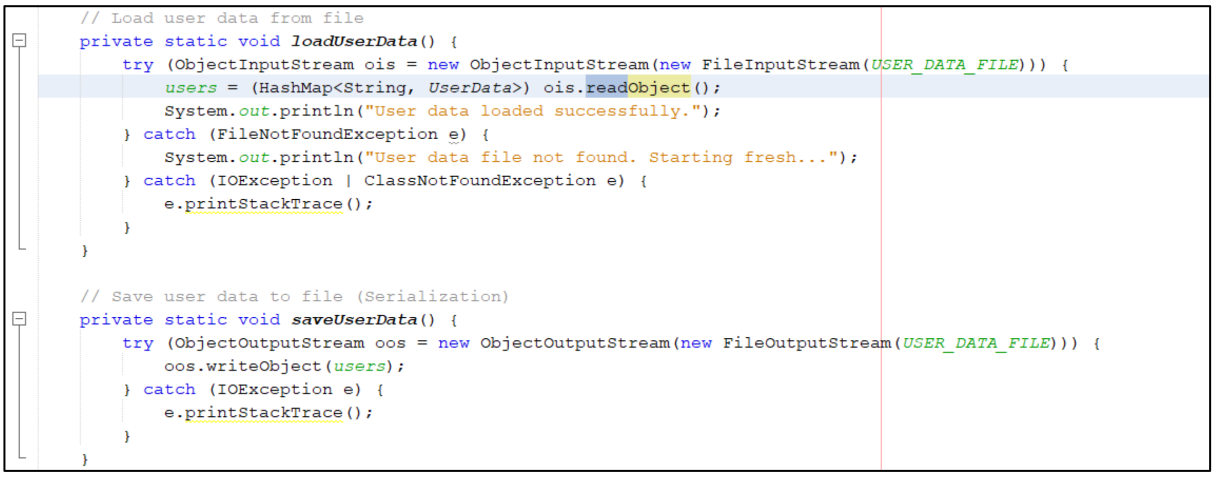
As mentioned before, serialization is an important java programming language that developers use to store and fetch object states efficiently. The use of serialization is widely recognized for mobile applications security such as forensics in keeping the data secure. Many types of applications depend on serialization as it is the easiest protection to implement into their code. The code basically converts an object into a storable format that is encrypted with different additional words not readable to the human eyes. The data persist within its own system and made to be centralised when using RMI to retrieve the data. Current trends uses serialization as a form of security, cause database storage is not as effective in keeping the data as plain text. (D. Pawlaszczyk , 2022)



```java
package assignment_ds;

import java.io.*;
import java.util.HashMap;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.ResultSet;


public class ManageLogin implements Serializable {
    private static final String USER_DATA_FILE = "users.ser";
    private static HashMap<String, UserData> users = new HashMap<>();
    private static int loggedInEmployeeID = -1; // To hold the ID of the logged-in employee

    static {
        loadUserData();
    }
```

Figure 2.1: Imports for Serialization

In our system, serialization is used to manage the login information of the employees through data transmission between systems. Remote Method Invocation (RMI) uses this method to secure the data transmission across the network as the simple call function of writeObject() and readObject() is used for this communication (A. Prakash , n.d). This mechanism uses a simple referencing method to find each of the objects within the text file.

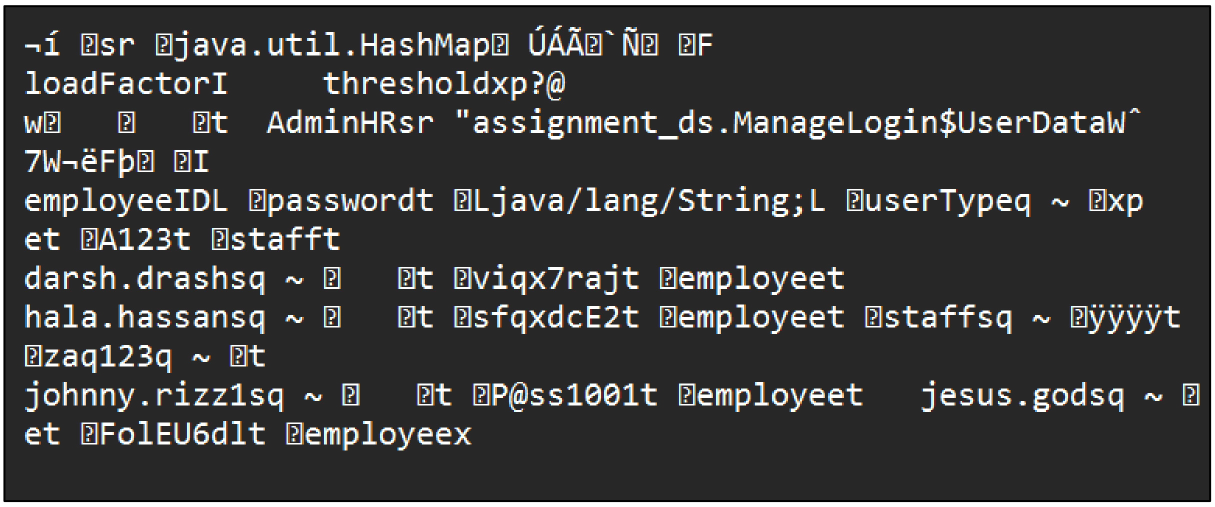


```
    // Load user data from file
    private static void loadUserData() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(USER_DATA_FILE))) {
            users = (HashMap<String, UserData>) ois.readObject();
            System.out.println("User data loaded successfully.");
        } catch (FileNotFoundException e) {
            System.out.println("User data file not found. Starting fresh...");
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    // Save user data to file (Serialization)
    private static void saveUserData() {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(USER_DATA_FILE))) {
            oos.writeObject(users);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Figure 2.2: Read & Write for Serialization

However, there is still vulnerability in serialization when the code is not properly written. There could be malicious attacks like unauthorized object manipulation or deserialization attacks. That is why ManageLogin in our system ensures that the user credential is stored efficiently by saving the data in HashMap. It is one of the serialization techniques to secure the essential data login. This caching approach improves login performance by avoiding the need for several database requests. During authentication, the system pulls previously saved credentials to verify login attempts.

Therefore, when the system restarts again the login credentials will be saved in its User.ser file, ensuring accessible even after the application is restarted. The data can only be manipulated from the system itself and altered in the User.ser file. In our program, maintaining backward compatibility ensures that previously stored user data is still accessible even when the class structure is updated. Below shows the example of how our file User.ser will look.



```
¬í sr java.util.HashMap ÚÁÃ`Ñ F
loadFactorI     thresholdxp?@
w     t  AdminHRsr "assignment_ds.ManageLogin$UserDataWˆ
7W¬ëFþ I
employeeIDL passwordt Ljava/lang/String;L userTypeq ~ xp
et A123t stafft
darsh.drashsq ~    t viqx7rajt employeet
hala.hassansq ~    t sfqxdcE2t employeet staffsq ~ ÿÿÿÿt
zaq123q ~ t
johnny.rizz1sq ~    t P@ss1001t employeet   jesus.godsq ~ 
et FolEU6dlt employeex
```

Figure 2.3: User.ser's Text file

To prevent risks such as malicious attacks or tampering the file for information, applications should use secure serialisation approaches such as whitelisting authorised classes during deserialization or, if necessary, alternate forms such as JSON or XML (D. Pawlaszczyk , 2022). Besides that, another issue with serialisation is when we append more information into the binary file, the file will produce larger memory space. This causes the performance to decrease and not suitable to run high efficiency applications. To support these type of situations, the application must use Protocol Buffers or specific serialisation algorithms to increase the speed performance while decreasing storage overhead (D. Pawlaszczyk , 2022).

In conclusion, the serialisation used in our project enables us to store, transmit, and retrieve object data effectively and precisely. Its used in our login management, remote procedure calls, and distributed applications where it is safe and secure to use.

7.2 ROLE OF MULTITHREADING

Multi-threading helps the HRM system perform simultaneous requests efficiently, allowing several employee and HR staff operations to be handled concurrently (Parsons, 2020). By implementing parallel execution threads, the system manages numerous tasks concurrently rather than sequentially, resulting in considerably quicker responses when multiple users interact with the system (Shukla, 2023). These independent execution paths enable effective utilization of system resources, with different threads handling profile updates, leave applications, and report generation simultaneously (Thomadakis et al., 2022). This approach is particularly beneficial for database operations because one thread can process updates while the others handle queries, preventing I/O bottlenecks and providing consistent performance during peak usage periods (Rashid et al., 2021).

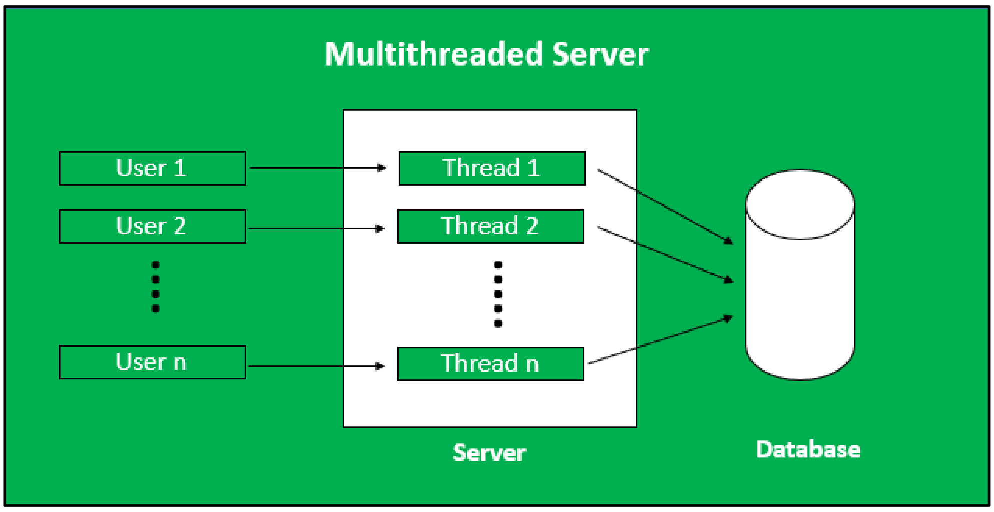
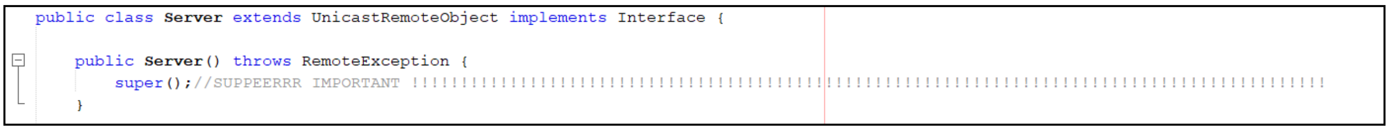


Figure 3.1: Example of Multithreaded Server (GeeksforGeeks, 2020)

The provided code demonstrates the application of multi-threading in the distributed HRM system:

1. Server-Side Multi-threading: Multi-threading is employed in the server component of the distributed HRM system to handle multiple client requests concurrently. The Server class extends the UnicastRemoteObject class and implements the Interface interface, which defines the remote methods that clients can invoke.

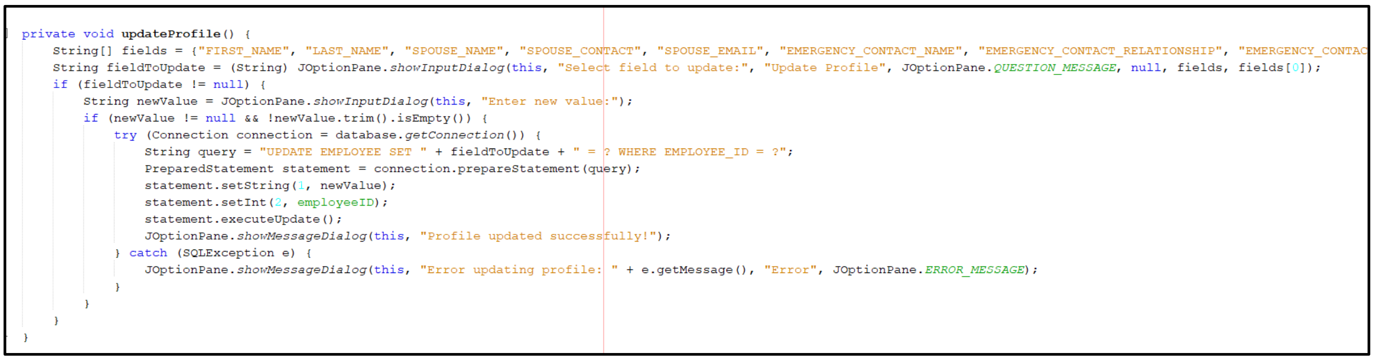


```
public class Server extends UnicastRemoteObject implements Interface {

    public Server() throws RemoteException {
        super();//SUPPEERRR IMPORTANT !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    }
```

Figure 3.2: Server-Side Multi-threading Code Snippet

As seen in figure 1.2, by extending UnicastRemoteObject, the server can automatically start a new thread for each request from the clients. This allows the server to process multiple requests simultaneously without blocking or waiting for each request to complete before moving to the next request. Multi-threading on the server side promotes efficient utilization of system resources and improves overall responsiveness.

2. Updating Employee Details: The updateProfile method in the employee dashboard side shows how multi-threading enables both the employee's profile and family details to be updated simultaneously.

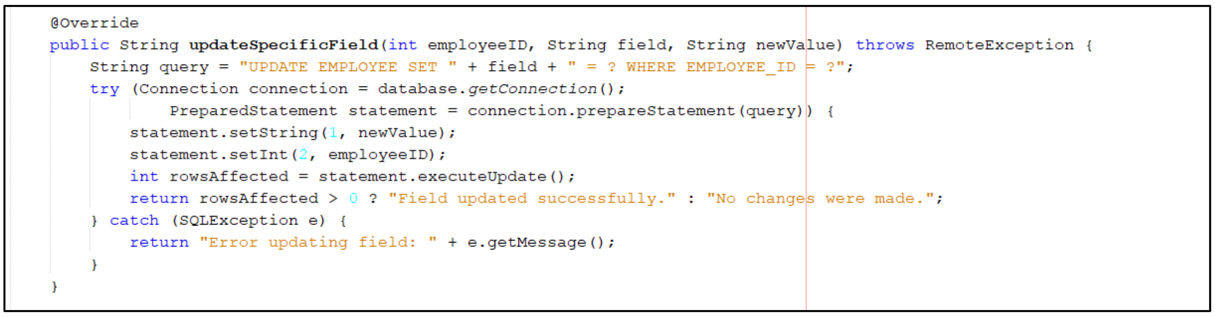


```
private void updateProfile() {
    String[] fields = {"FIRST_NAME", "LAST_NAME", "SPOUSE_NAME", "SPOUSE_CONTACT", "SPOUSE_EMAIL", "EMERGENCY_CONTACT_NAME", "EMERGENCY_CONTACT_RELATIONSHIP", "EMERGENCY_CONTAC
    String fieldToUpdate = (String) JOptionPane.showInputDialog(this, "Select field to update:", "Update Profile", JOptionPane.QUESTION_MESSAGE, null, fields, fields[0]);
    if (fieldToUpdate != null) {
        String newValue = JOptionPane.showInputDialog(this, "Enter new value:");
        if (newValue != null && !newValue.trim().isEmpty()) {
            try (Connection connection = database.getConnection()) {
                String query = "UPDATE EMPLOYEE SET " + fieldToUpdate + " = ? WHERE EMPLOYEE_ID = ?";
                PreparedStatement statement = connection.prepareStatement(query);
                statement.setString(1, newValue);
                statement.setInt(2, employeeID);
                statement.executeUpdate();
                JOptionPane.showMessageDialog(this, "Profile updated successfully!");
            } catch (SQLException e) {
                JOptionPane.showMessageDialog(this, "Error updating profile: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}
```

Figure 3.3: Update Profile Code Snippet

As shown in figure 1.3, The updateProfile method illustrates the use of multi-threading in the case of updating employee profiles. When several employees at the same time try to update their profiles using this method, each request for update is processed in an independent thread so that they can be executed concurrently. The method opens a database connection, builds an SQL update query depending on the chosen field, and updates using a prepared statement. Multi-threading allows efficient processing of a large volume of profile update requests, improving the HRM system's performance and responsiveness as well as data consistency through proper database management and error handling.

At the server side, the updateSpecificField method does updating in the database as well.

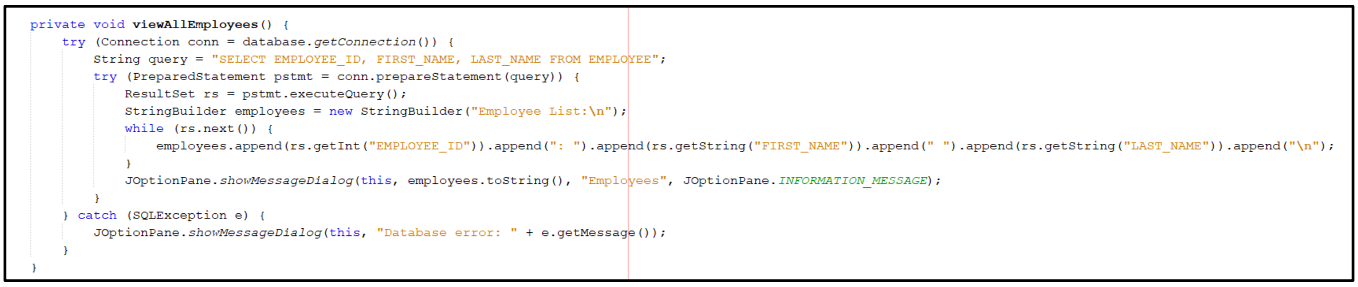


```java
@Override
public String updateSpecificField(int employeeID, String field, String newValue) throws RemoteException {
    String query = "UPDATE EMPLOYEE SET " + field + " = ? WHERE EMPLOYEE_ID = ?";
    try (Connection connection = database.getConnection();
            PreparedStatement statement = connection.prepareStatement(query)) {
        statement.setString(1, newValue);
        statement.setInt(2, employeeID);
        int rowsAffected = statement.executeUpdate();
        return rowsAffected > 0 ? "Field updated successfully." : "No changes were made.";
    } catch (SQLException e) {
        return "Error updating field: " + e.getMessage();
    }
}
```

Figure 3.4: Update Specific Field Code Snippet

In figure 1.4, The updateSpecificField method executes the update query using a prepared statement. Multiple threads may call this method simultaneously, and the database management system handles concurrent access and preserves data consistency. Multi-threading at this level allows for effective processing of update requests from multiple employees.

3. Viewing Employee Information: Multi-threading enables staff to view the most recent modifications done by employees in real-time. If an employee updates his or her profile or family data, the data is reflected in the database immediately. For the staff to be able to view these updates, the system must retrieve the most current information from the database whenever staff view employee information.



```java
private void viewAllEmployees() {
    try (Connection conn = database.getConnection()) {
        String query = "SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME FROM EMPLOYEE";
        try (PreparedStatement pstmt = conn.prepareStatement(query)) {
            ResultSet rs = pstmt.executeQuery();
            StringBuilder employees = new StringBuilder("Employee List:\n");
            while (rs.next()) {
                employees.append(rs.getInt("EMPLOYEE_ID")).append(": ").append(rs.getString("FIRST_NAME")).append(" ").append(rs.getString("LAST_NAME")).append("\n");
            }
            JOptionPane.showMessageDialog(this, employees.toString(), "Employees", JOptionPane.INFORMATION_MESSAGE);
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Database error: " + e.getMessage());
    }
}
```

Figure 3.5: View All Employees Code Snippet

As presented in figure 1.5, in the viewAllEmployees method, the staff dashboard retrieves the list of employees from the database. The method executes a SELECT query to retrieve the employee IDs, first names, and last names from the EMPLOYEE table. By executing this query each time the staff views the list of employees, they always receive the latest data, including any changes performed by the employees through their profile updates.

7.3 ROLE OF OBJECTED ORIENTED PROGRAMMING

Object-Oriented Programming (OOP) transformed software system development, particularly for distributed computing. OOP organizes software around objects—class instances containing data and behavior (Singh et al., 2021). OOP develops maintainable, extendable, and modular distributed systems.

The core OOP principles—encapsulation, inheritance, and polymorphism—solve distributed system problems effectively. Encapsulation bundles data and functions within objects, ensuring data integrity and reducing dependencies. Inheritance enables specialized classes to be derived from existing ones, promoting code reuse. Polymorphism enables objects of different classes to be treated as objects of a common parent class, promoting loose coupling and design flexibility (Nagineni, 2021).

OOP also facilitates separation of concerns, breaking complex systems into smaller objects with well-defined responsibilities. This makes development more manageable, enhances code readability, and facilitates team collaboration. It further enables creation of reusable components that can be shared across the distributed system, lowering development effort and improving effectiveness (Nagineni, 2021).

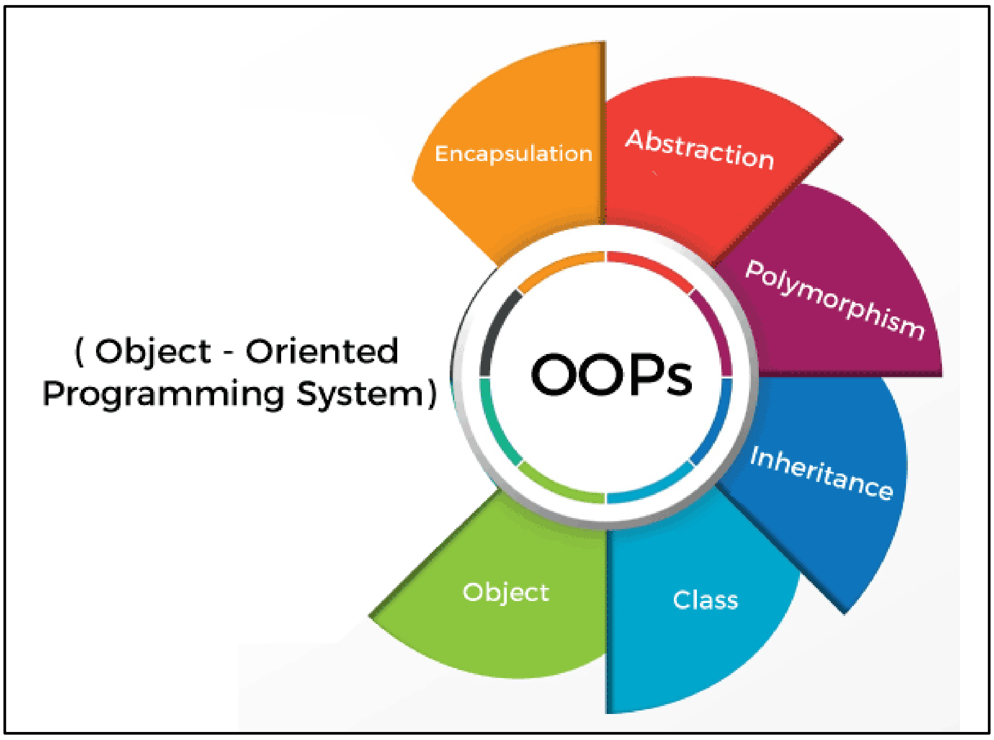
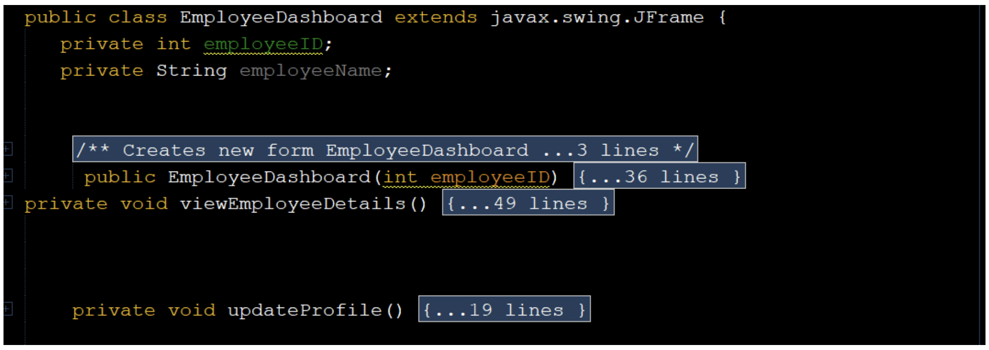


Figure 4.1: Principles of OOP (Akalanka, 2023)

ENCAPSULATION

Encapsulation is one of the key principles of OOP that binds data and functions together within a class, hiding the internal processes and providing a public interface for interaction (Singh et al., 2021). Encapsulation is observable in the design of various classes in the provided code.



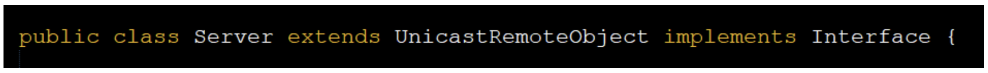
```
public class EmployeeDashboard extends javax.swing.JFrame {
   private int employeeID;
   private String employeeName;


     /** Creates new form EmployeeDashboard ...3 lines */
      public EmployeeDashboard(int employeeID) {...36 lines }
private void viewEmployeeDetails() {...49 lines }



    private void updateProfile() {...19 lines }
```

Figure 4.2: Encapsulation in Employee Dashboard

In figure 2.2, the EmployeeDashboard class encapsulates the behavior and data related to the employee dashboard functionality. The class has a private employeeID field, which is accessible only within the class. The class has methods like viewEmployeeDetails() and updateProfile() that encapsulate the logic for viewing employee details and updating profiles, respectively. Encapsulation helps in the achieving of data integrity and reduces dependencies between classes (Gupta & Sharma, 2022).

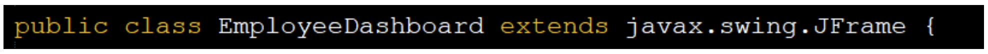
INHERITANCE

Inheritance: Inheritance is an OOP concept where a class acquires characteristics and functions of another class, which enables code reuse and hierarchical organization (Nagineni, 2021). In the provided code, inheritance has been used to expand the functionality of already existing classes.



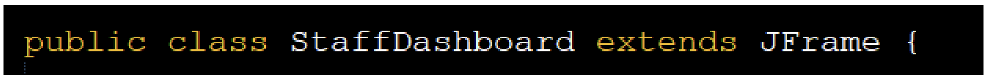
```
public class Server extends UnicastRemoteObject implements Interface {
```

Figure 4.3: Inheritance in Server



```
public class EmployeeDashboard extends javax.swing.JFrame {
```

Figure 4.4: Inheritance in Employee Dashboard



```
public class StaffDashboard extends JFrame {
```

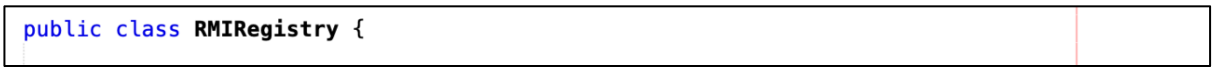
Figure 4.5: Inheritance in Staff Dashboard

As presented in figures 2.3, 2.4, & 2.5, the Server class has inherited remote object communication functionality from the UnicastRemoteObject class. Similarly, the EmployeeDashboard and StaffDashboard classes have inherited Java Swing frame's properties and methods from the javax.swing.JFrameclass. Inheritance makes it possible to create specialized classes from the pre-existing ones, promoting code reuse and modularity.

MODULARITY

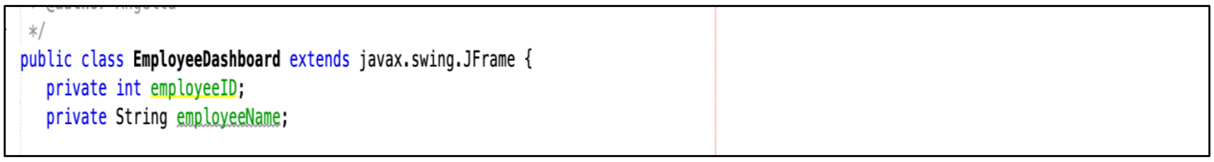
Modularity refers to the practice of dividing a software into various components which can work together to form a single functioning item, or they can work independently if they are not connected to each other. Modularity helps in reducing the complexity of software programs, enhance code reusability and allows for developers to perform unit and integration testing to test functionalities while the system is still undergoing development (Rohan Vats, 2022).

In our system, we have implemented modularity by using classes and methods.



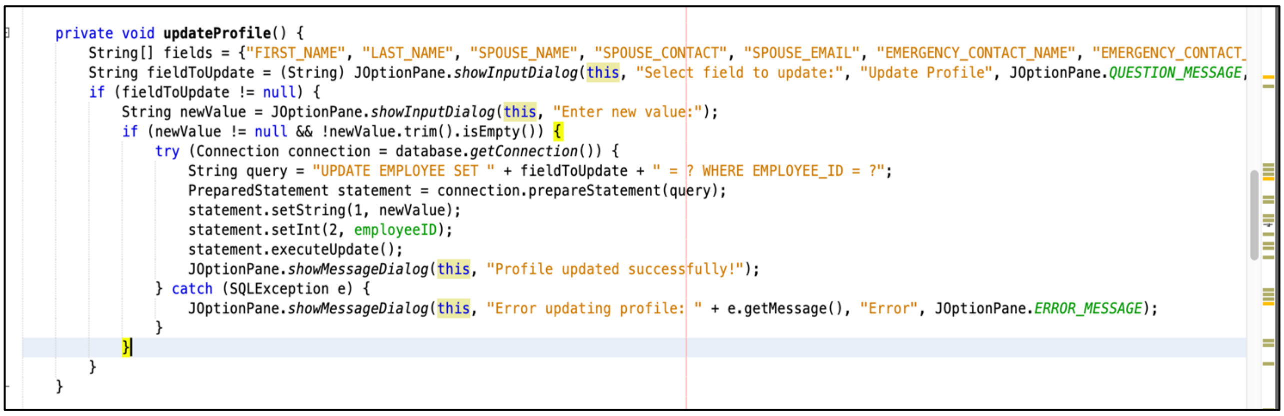
```
public class RMIRegistry {
```

Figure 5.1: RMI Registry Class



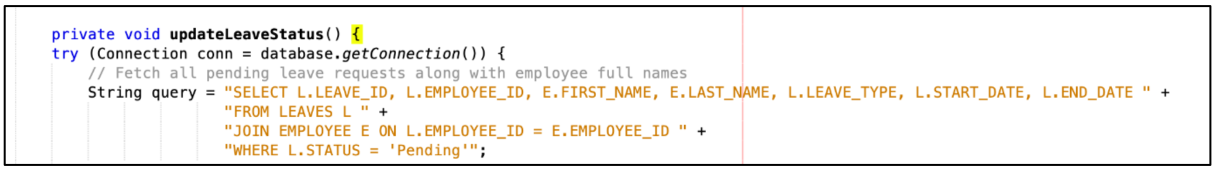
```
 */
public class EmployeeDashboard extends javax.swing.JFrame {
    private int employeeID;
    private String employeeName;
```

Figure 5.2: Employee Dashboard Class



```
    private void updateProfile() {
        String[] fields = {"FIRST_NAME", "LAST_NAME", "SPOUSE_NAME", "SPOUSE_CONTACT", "SPOUSE_EMAIL", "EMERGENCY_CONTACT_NAME", "EMERGENCY_CONTACT_
        String fieldToUpdate = (String) JOptionPane.showInputDialog(this, "Select field to update:", "Update Profile", JOptionPane.QUESTION_MESSAGE,
        if (fieldToUpdate != null) {
            String newValue = JOptionPane.showInputDialog(this, "Enter new value:");
            if (newValue != null && !newValue.trim().isEmpty()) {
                try (Connection connection = database.getConnection()) {
                    String query = "UPDATE EMPLOYEE SET " + fieldToUpdate + " = ? WHERE EMPLOYEE_ID = ?";
                    PreparedStatement statement = connection.prepareStatement(query);
                    statement.setString(1, newValue);
                    statement.setInt(2, employeeID);
                    statement.executeUpdate();
                    JOptionPane.showMessageDialog(this, "Profile updated successfully!");
                } catch (SQLException e) {
                    JOptionPane.showMessageDialog(this, "Error updating profile: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
    }
```

Figure 5.3: Method in Employee Dashboard



```
    private void updateLeaveStatus() {
    try (Connection conn = database.getConnection()) {
        // Fetch all pending leave requests along with employee full names
        String query = "SELECT L.LEAVE_ID, L.EMPLOYEE_ID, E.FIRST_NAME, E.LAST_NAME, L.LEAVE_TYPE, L.START_DATE, L.END_DATE " +
                       "FROM LEAVES L " +
                       "JOIN EMPLOYEE E ON L.EMPLOYEE_ID = E.EMPLOYEE_ID " +
                       "WHERE L.STATUS = 'Pending'";
```

Figure 5.3: Method in Staff Dashboard

Above are some examples of the well-structured classes and methods that have been implemented in our system for modularity purposes, The above methods can be reused and make the system more organized, scalable and easier to maintain.

8.0 USE CASE DIAGRAM

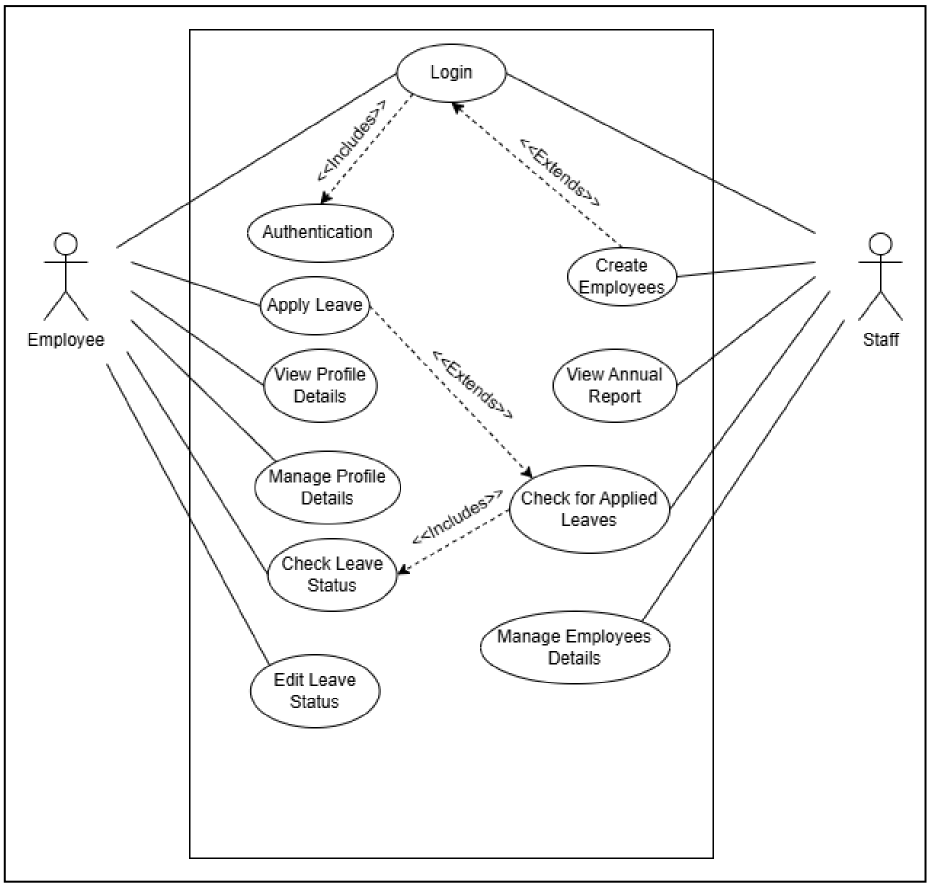


Figure 6.1: Use Case Diagram of HRM System

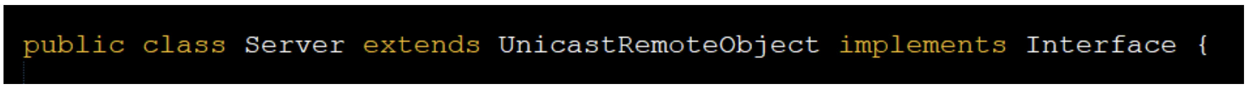
Employees can:	Staff can:
○ Login (which includes authentication). ○ Apply for leave. ○ View and manage profile details. ○ Check leave status. ○ Edit leave status.	○ Login. ○ Create employees. ○ View annual reports. ○ Check for applied leaves (which includes checking leave status). ○ Manage employee details.

9.0 PROTOCOLS

9.1 JAVA REMOTE METHOD PROTOCOL (JRMP)

The HRM system implements JRMP for distributed communication between components. JRMP facilitates remote method invocation and object serialization within the system (Qu, 2021).

On the server side, the Server class extends UnicastRemoteObject and implements the remote interface Interface.



```
public class Server extends UnicastRemoteObject implements Interface {
```

Figure 7.1: JRMP in Server

Implementation Breakdown:

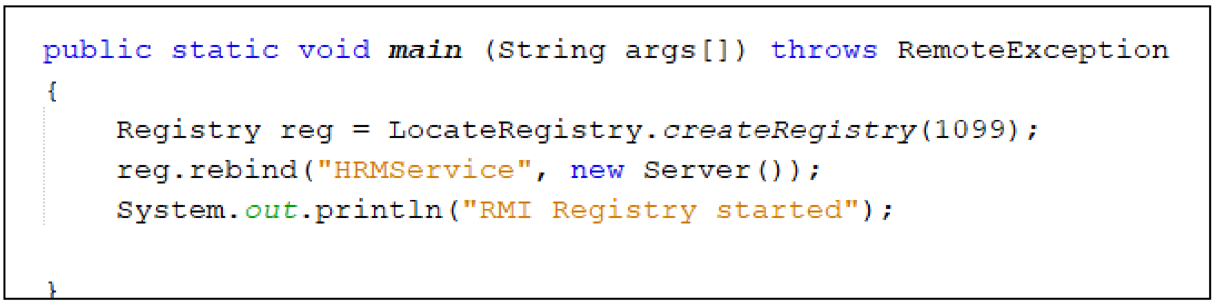
- The Server class extends UnicastRemoteObject, which automatically uses JRMP to handle remote method invocation and serialize/deserialize objects between the client and server components
- JRMP activation requires no additional configuration due to this inheritance pattern
- By implementing Interface, the server defines which methods are accessible remotely to clients.

9.2 TRANSMISSION CONTROL PROTOCOL (TCP)

The HRM system utilizes Transmission Control Protocol to ensure that the communication between the client, RMI server, and database is reliable. Since RMI depends on TCP(JRMP), it ensures secure method invocation and maintains data integrity.

TCP in RMI communication:

The RMI registry operates on TCP port 1099, allowing client to locate and invoke remote methods. When employees submit leave requests, the client connects to the RMI service over TCP, which is shown below.



```
public static void main (String args[]) throws RemoteException
{
    Registry reg = LocateRegistry.createRegistry(1099);
    reg.rebind("HRMService", new Server());
    System.out.println("RMI Registry started");

}
```

Figure 7.2: TCP use in RMI Registry

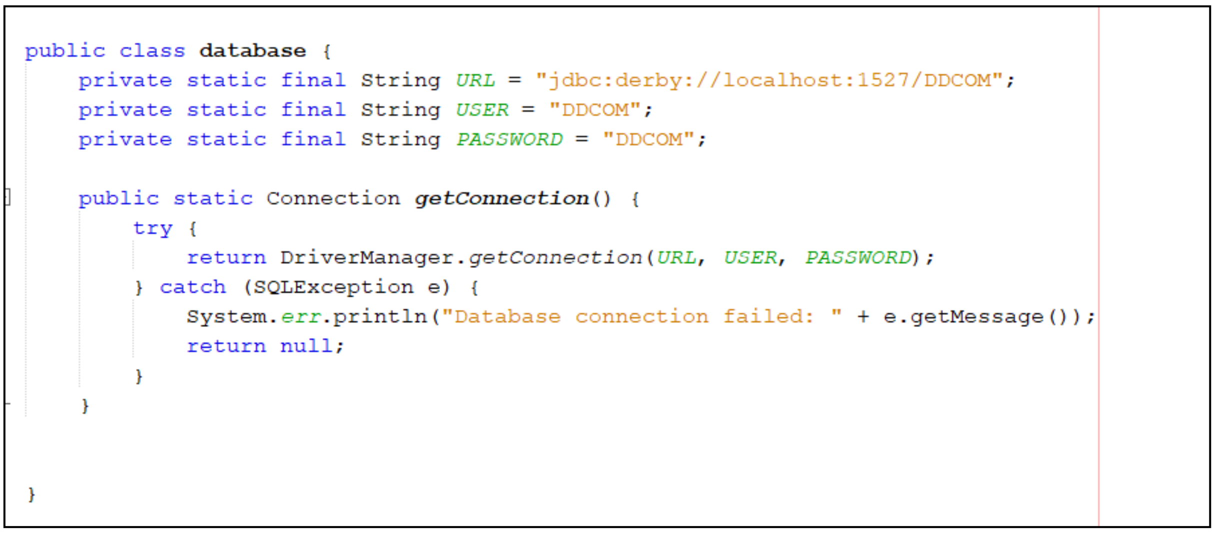
Implementation Breakdown:

- The RMI registry runs on TCP port 1099 through **LocateRegistry.createRegistry(1099),** enabling client to locate services.

- The **HRMService** is bound to the registry through **reg.rebind("HRMService", new Server ())** enabling remote method invocation over TCP.

TCP in Database Connectivity:

The Java Derby database runs on TCP port 1527, allowing persistent and secure transactions which has been depicted below.



```java
public class database {
    private static final String URL = "jdbc:derby://localhost:1527/DDCOM";
    private static final String USER = "DDCOM";
    private static final String PASSWORD = "DDCOM";

    public static Connection getConnection() {
        try {
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (SQLException e) {
            System.err.println("Database connection failed: " + e.getMessage());
            return null;
        }
    }


}
```

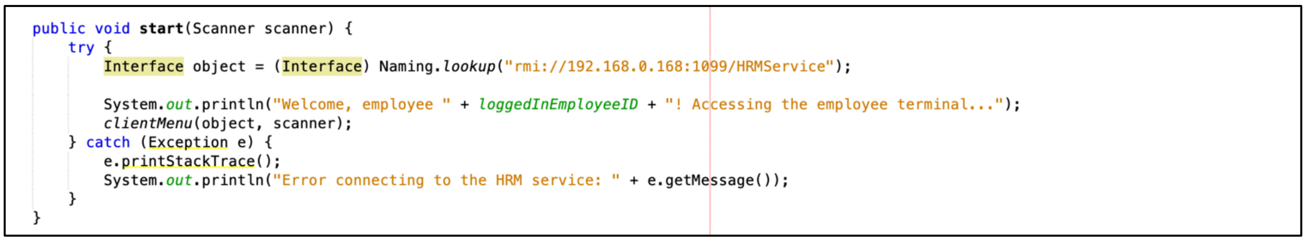
Figure 7.2: JDBC in TCP

Implementation Breakdown:

- The **URL "jdbc: derby://localhost:1527//DDCOM"** verifies that Derby operates on TCP port 1527.
- **DriveManager.getConnection(URL, USER, PASSWORD);** creates a secure and persistent connection.
- The **try-catch** block manages failures to guarantee reliability.

9.3 INTERNET PROTOCOL (IP)

The HRM system uses the IP protocol for communication as it is responsible for delivering packets from the source to a destination across interconnected networks. The version of IP that the HRM system uses is the IPv4 where it specifies the IPv4 address of the server (destination) in the Client side so the client can access the server and request services from the server.



```
public void start(Scanner scanner) {
    try {
        Interface object = (Interface) Naming.lookup("rmi://192.168.0.168:1099/HRMService");

        System.out.println("Welcome, employee " + loggedInEmployeeID + "! Accessing the employee terminal...");
        clientMenu(object, scanner);
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("Error connecting to the HRM service: " + e.getMessage());
    }
}
```

Figure 7.3: Implementation on Employee Side

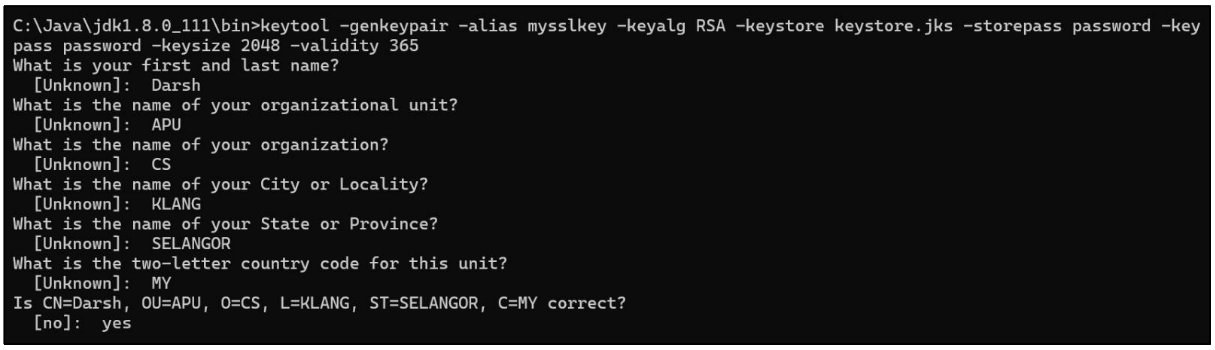
From the above figure, we can see that the employee side looks up the IP address of the server **(Interface object = (Interface) Naming.lookup("rmi://192.168.0.168:1099/HRMService"))** on port 1099 so it can establish a connection and access the employee terminal.

9.4 SECURE SOCKET LAYEER (SSL)

The SSL protocol is used in the HRM system to provide secure communication over the network, it ensures data integrity, confidentiality and authentication when transmitting the data. SSL is widely used in distributed computing systems as a cryptographic protocol.

```
þíþí   ▯   ▯   ▯ ▯mysslkey  ▯•EœIÍ  ▯▯0,▯ý0▯▯
+▯▯▯▯*▯▯▯▯▯ ▯,▯é^Ê▯-▯Õ7c4È▯û®û9˜‡▯▯t▯9▯▯ÐP_▯ÎùpŽ/M2
Í▯▯ÿh▯▯▯▯▯kñ▯
‹Ì kñTÏ–▯² Ã€d«Ó        ¬–Ñ!sÓ<#p·õ|ºõ/ÙféÙV@-Í…2å(▯Ç¤ì£TG'▯øÁ7▯▯?ó·®MºÊÄ▯1<ñÜ“f¬± :~wÚÙÓùé¶´×äp▯Õ†ñ;û„æßµyœ5}▯ˆßë)%…W▯▯Z‰\‘uw¥%J»Â+(þ%õïUëšún▯âñf/
ç8ò▯▯fSÆÍß¥fX)Úb|7-|QÐÁ▯÷Wù¦–▯F10▯▯.Ø”c´¡ò'ü–ß8St)Ÿ‹êÐ▯*AÒ.&Xp▯£<▯ãù’0š}¬f¨Ã½±ì·(Éñ _▯^¶▯Ó'Y÷Îy·▯▯”ÒŸ–▯]▯Ô¦#ûoºŸÓ™Ùè▯!²Œº
§:ôN▯¶pß▯▯▯›▯æÔëõK©:ŽžDgs±▯QúÔ/ Ì
WÖþÚ
L▯E▯▯BÌóûòµ`óxÈ2Ü8xá▯ö^ìV[Ø3Ôd@å▯ë▯IÁ€%,▯▯▯ï▯T–rf¶▯ª‡▯,¥¨“Ól>·Ç▯▯´zÔ7¤Àè'‹PùwÌ@#˜ÎžC    wÑ▯Ó▯EÇíÌ>Ê▯>÷▯ÍðŽb%Xã▯2û1p1<1¤#ëµ‡M▯▯Ä±íUÑ‡ý0é▯”¢.ÿŒÿºY§ÿ+Õ▯ùB©„▯
\▯´▯5È▯ŸkÂ¤Ž¹„®ŽÉ¬▯UÈŒ-öS▯©¹iå&▯¢†’_özS£%sB")È8Ù_‹▯„€Ñb?ž¿G‡çÜ0{E▯)@;àóÕ▯÷û3¶©×®…¦7.6‰▯T`â‡’ÞI%Ø▯ÿÍÓJŸ¹▯S=>ž▯ Yž†ƒ]SsX?³
       ÅQÎÜ6▯‘Lòïk▯▯k▯Þ9R=▯▯DÄ6Ô“EnÆÁ▯¤ºdúÛô;èëà▯ö2¤]•KÛ       UN-më
O8^¥
0ß▯]1ØÜ²(4▯¬VÓ▯hbJ›ö▯»áŒÎ▯7▯9+ˆÑJ\Þä▯ŠxÝo²;ë¡uÞR1▯Ÿé‡’+ ò>©ñ^j>Ó]T’neššgªyœÆn=õ¿òc2æ▯@À-
mÇ¦^ùè▯¿ÒA+]ç▯▯aPìž▯▯ð#:âE%x¹▯{[/RwDo‡cC^ÅO▯▯KD<ÓqŸÉu‰,▯▯▯,dÉq•åæK›öàÁ+Š▯,ù▯+9Õá▯aº¿aþ*…PÛ´ž    N▯í▯ïä„‡³˜‰ÌýôEó«▯]▯™„ý▯€þÏ%NSMº,¨9ækõÃ·çæéu!▯▯š▯-2
±%m[…=G“RãH     ▯£Xž14▯[ÑB»É▯▯¥¡·h(Ò▯žnG¦˜▯Ý*ù;AÉÁ¶
gç ▯w Kçþûnõ–▯▯G§ÿjpXÛ&ç▯þéÄrcž¤O™ào▯ýk !wÆ-Îk&˜G‡,E%€gC}%›Ì•Ë
¦[u▯▯T▯».f▯¨▯Ó;▯▯Ì•ïÈ)▯‡4Þ5F•`ÜÖ[V ·Yí-»%▯p5º▯×ÕÒ{Šæ1¥<œ▯ÿ²%LÜ•ä„#Z
ær×ô¨VLm"ÌÞI▯èÔì▯5▯!»—M]ÆO%Æö▯ÏNä▯ç¤▯'˜¥š{Íå▯Ö
µRsr▯,÷
âáã©Ó3ÅÍ?ºg▯·ÞU”º†▯˜É’ ç‘ì‡áêÕ▯€▯+▯Öpu▯8| 3íVí▯Ä&íÓ1.º¬pâãE åæù¨ 4Ù% „▯
Àóëþv(œZDÒb:û˜ØåÚ˜▯ž&▯#¶ƒ®Õë"   ▯ ▯X.509  ▯O0,▯K0,▯3 ▯▯▯▯▯▯*Ì▯®0
▯       *†H†÷
▯▯
▯ 0V1
0       ▯▯U▯▯▯▯MY1
0
▯▯U▯▯▯▯Sel1▯0
▯▯U▯ ▯▯Klang1
0       ▯▯U▯
▯▯CS1
0
▯▯U▯
▯▯APU1▯0
▯▯U▯▯▯▯darsh0▯▯
```

Figure 7.4: Java Keystore Certificate



```
C:\Java\jdk1.8.0_111\bin>keytool -genkeypair -alias mysslkey -keyalg RSA -keystore keystore.jks -storepass password -key
pass password -keysize 2048 -validity 365
What is your first and last name?
  [Unknown]:  Darsh
What is the name of your organizational unit?
  [Unknown]:  APU
What is the name of your organization?
  [Unknown]:  CS
What is the name of your City or Locality?
  [Unknown]:  KLANG
What is the name of your State or Province?
  [Unknown]:  SELANGOR
What is the two-letter country code for this unit?
  [Unknown]:  MY
Is CN=Darsh, OU=APU, O=CS, L=KLANG, ST=SELANGOR, C=MY correct?
  [no]:  yes
```

Figure 7.5: SSL Encryption Process

The figure 7.4 shows the self-signed certificate "keystore.jks" that was generated after performing the SSL encryption and is used to secure the RMI (Remote Method Invocation) file. Figure 7.5 shows the entries which are used to generate the private public keypair (keystore.jks") used for encryption and decryption.

10.0 CONCLUSION

Implementing the Human Resources Management (HRM) system utilizing Java Remote Method Invocation (RMI) successfully modernizes PENTAFOUR old manual leave management system by resolving inefficiencies and errors linked to paper-based practices. The system ensures immediate communication between employees and Human Resources (HR) departments, offering essential features like the management of employee profiles, leave submission, leave status, and detailed report creation. The utilization of multi-threading, serialization and Object-Oriented Programming (OOP) concepts has enhanced the system's maintainability along with scalability within a distributed setting. Integration of a centralized database through Derby ensures employee record protection and management structure. All the specified features and applications of methods have provided the best solution for PENTAFOUR leave management system.

11.0 FUTURE ENHANCEMENT

The existing HRM system effectively deals with leave requests and employee records but introducing further improvements will surely optimize its performance, data security and scalability. Cloud integration can be considered a significant advancement for improved accessibility, scalability and overall system reliability. Real-time data updates through the existence of cloud integration ensure leave submission and approvals across every Human Resources (HR) department are seamless. The integration of AI-driven analytics would offer predictive insights regarding employee leave behaviour by helping Human Resources (HR) departments make data-informed decisions for better workforce management. A mobile responsive design optimization would improve accessibility allowing employees to conveniently apply for leave and check status despite their locations. Lastly, incorporating two-factor authentication (2FA) to secure access and introduce automated email alerts would significantly improve user experience, security and system performance.

12.0 REFERENCES

Kambur, E., & Yildirim, T. (2022). From traditional to smart human resources management. *International Journal of Manpower*, *44*(3), 422–452. https://doi.org/10.1108/ijm-10-2021-0622

Taslim, W. S., Rosnani, T., & Fauzan, R. (2024). The impact of work automation on human resource decision making: The mediating role of employee performance. *International Journal of Social Science and Human Research*, *07*(12). https://doi.org/10.47191/ijsshr/v7-i12-46

Marín, J. M. M., De Oliveira-Dias, D., Navimipour, N. J., Gardas, B., & Unal, M. (2021). Cloud computing and human resource management: systematic literature review and future research agenda. *Kybernetes*, *51*(6), 2172–2191. https://doi.org/10.1108/k-05-2021-0420

Kumar, Penki & Tirumala, Pentapati. (2022). Comparative Study of HR Management Practices in Information Technology Industry. 14. 2022. 10.9756/INTJECSE/V14I5.175.

Epen, S., Jayalakshmi, M., Ravi Kumar, Penki., & Tirumala, P. (2022). Comparative Study of HR Management Practices in Information Technology Industry. *International Journal of Early Childhood Special Education (INT-JECSE)*, *14*(05). https://doi.org/10.9756/INTJECSE/V14I5.175

Sonar, N. A., & Pandey, N. D. R. K. (2023). Human Resource (HR) Practices - A Comprehensive review. *Management Journal for Advanced Research*, *3*(5), 42–56. https://doi.org/10.54741/mjar.3.5.5

Abuhantash, A. (2023). The Impact of Human resource information Systems on Organizational Performance: A Systematic literature review. European Journal of Business Management and Research, 8(3), 239–245. https://doi.org/10.24018/ejbmr.2023.8.3.1992

Destriani, R., Adhitama, R. Y., Sensuse, D. I., Hidayat, D. S., & Purwaningsih, E. H. (2024). Challenges and Technology Trends in Implementing a Human Resource Management System: A Systematic Literature review. *Journal of Information Systems Engineering and Business Intelligence*, *10*(3), 355–367. https://doi.org/10.20473/jisebi.10.3.355-367

Porkodi, S., & Raman, A. M. (2024). Success of cloud computing adoption over an era in human resource management systems: a comprehensive meta-analytic literature review. *Management Review Quarterly*. https://doi.org/10.1007/s11301-023-00401-0

Parsons, D. (2020). Multithreading. In *Texts in computer science* (pp. 405–436). https://doi.org/10.1007/978-3-030-54518-5_16

Thomadakis, P., Tsolakis, C., & Chrisochoides, N. (2022). Multithreaded runtime framework for parallel and adaptive applications. *Engineering With Computers*, *38*(5), 4675–4695. https://doi.org/10.1007/s00366-022-01713-7

Rashid, Z. N., Zeebaree, S. R. M., Zebari, R. R., Ahmed, S. H., Shukur, H. M., & Alkhayyat, A. (2021). Distributed and Parallel Computing System Using Single-Client Multi-Hash Multi-Server Multi-Thread. *2021 1st Babylon International Conference on Information Technology and Science (BICITS)*, 222–227. https://doi.org/10.1109/bicits51482.2021.9509872

GeeksforGeeks. (2020, November 9). *Multithreaded servers in Java*. GeeksforGeeks. https://www.geeksforgeeks.org/multithreaded-servers-in-java/

Singh, N., Chouhan, S. S., & Verma, K. (2021). Object Oriented Programming: concepts, limitations and application trends. *2021 5th International Conference on Information Systems and Computer Networks (ISCON)*, 1–4. https://doi.org/10.1109/iscon52037.2021.9702463

Nagineni, R. (2021). A research on object oriented programming and its concepts. *International Journal of Advanced Trends in Computer Science and Engineering*, *10*(2), 746–749. https://doi.org/10.30534/ijatcse/2021/401022021

Akalanka, R. (2023, January 6). Object Oriented Programming(OOP) in Java - ravindu Akalanka - Medium. *Medium*. https://medium.com/@ravinduakalankazoysa/object-oriented-programming-oop-in-java-8c3c219c294b

Qu, X. (2021). Application of Java Technology in dynamic web database technology. *Journal of Physics Conference Series*, *1744*(4), 042029. https://doi.org/10.1088/1742-6596/1744/4/042029

S. Horstmann, C. (2023). *Core Java, Volume I: Fundamentals* (13th ed., Vol. 1). Oracle Press. https://books.google.com.my/books?hl=en&lr=&id=TjYREQAAQBAJ&oi=fnd&pg=PT10&dq=java&ots=WC-0VBXiMt&sig=mpsy7b48i4flil9U6gW3pvV0VXU&redir_esc=y#v=onepage&q=java&f=false

Melanie. (2023, November 30). *CORBA: Infrastructure definition and benefits*. Data Science Courses | DataScientest. https://datascientest.com/en/corba-infrastructure-definition-and-benefits

What is SOAP API: Formats, Protocols, and Architecture. (n.d.). AltexSoft. https://www.altexsoft.com/blog/what-is-soap-formats-protocols-message-structure-and-how-soap-is-different-from-rest/

Basics of SOAP - Simple Object Access Protocol. (2019, September 27). GeeksforGeeks. https://www.geeksforgeeks.org/basics-of-soap-simple-object-access-protocol/

Mihailescu, M. (2015, March). *(PDF) Remote method invocation (RMI)*. Research Gate. https://www.researchgate.net/publication/275971083_Remote_Method_Invocation_RMI

Whims, S. (n.d.). *The component object model*. Win32 apps | Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/com/the-component-object-model

Opyrchal, L., & Prakash, A. (n.d.). Efficient object serialization in Java. Proceedings. 19th IEEE International Conference on Distributed Computing Systems. Workshops on Electronic Commerce and Web-Based Applications. Middleware. doi:10.1109/ecmdd.1999.776421

Pawlaszczyk, D. (2022, May). *(PDF) java serialization*. Research Gate. https://www.researchgate.net/publication/360354146_Java_Serialization

Rohan Vats. (2022, October). *Modularity in Java Explained With Step by Step Example [2024]*. UpGrad Blog. https://www.upgrad.com/blog/modularity-in-java/#

13.0 APPENDICES AND WORKLOAD MATRIX

NAME/ TP NUMBER	WORKLOAD	EXTRA REQUIREMENTS	SIGNATURE
HABEBA MOHAMED ABDELSALAM MOHAMED HASSAN (TP067863)	Abstract, Introduction, Problem Background, Requirements, Multithreading, Object-Oriented Programming (Encapsulation, Inheritance), Protocols (JRMP)	Document Formatting	[illegible]
DARSHRNNEY A/P PUSHPANATHAN (TP072253)	Research and Evaluation (RMI, DCOMS), Serialization , Use Case Diagram	Management of team, Document consistency	
ANGELITA DEBORAH A/P CHARLES TERRENCE LOURDES	Protocols (TCP), Conclusion, Future Enhancements		
LUBAINA ABDULLAH KHAN	Protocols (IP, SSL), Gantt Chart, Research and Evaluation (SOAP, CORBAs	Document Formatting	

WORD COUNT AT THE END OF THE REPORT (EXCLUDING TITLE, SOURCE CODE OF PROGRAM & CONTENTS PAGES):

The current word count excluding cover page, table of contents, references, and appendices is 4,504 words.