**Human Resources System  
Design Document**

**Submitted by**

Andulrahman Al-Awadhi

**Student ID**

2295404

**Vanier College**

Contents

[**Abstract** 2](#_Toc131019084)

[Introduction 3](#_Toc131019085)

[Assertion 3](#_Toc131019086)

[Functional Requirements 3](#_Toc131019087)

[Non-Functional Requirements 5](#_Toc131019088)

[UML Class Diagram 7](#_Toc131019089)

[Pseudo Code: 9](#_Toc131019090)

# **Abstract**

This paper presents the design and implementation of a software system that manages the payroll and administration of a university's teaching and non-teaching staff. The system comprises abstract and concrete classes, interfaces, custom exception handling, and file handling to provide a comprehensive solution to manage employee data, compute payroll, and maintain departmental structure within the university. The implementation includes classes for Person, Teacher, Staff, Department, and an interface for PayRoll. The system also utilizes exception handling to ensure data integrity and validity. Additionally, the implementation incorporates file handling to store and retrieve data, making it persistent and allowing for efficient updating of the information.

# Introduction

Colleges require efficient and reliable management systems to handle the administration of their staff members, including payroll calculations and departmental organization. This paper introduces a software system that meets these requirements by employing object-oriented programming principles to implement a comprehensive solution. The system consists of abstract and concrete classes, interfaces, and custom exception handling to ensure data integrity and validity. Additionally, the implementation uses file handling to store and retrieve data, making it persistent and allowing for efficient updating of the information.

# Assertion

The proposed system is a robust and efficient solution for managing university staff and payroll calculations, with the flexibility to accommodate different categories of employees, such as full-time and part-time teachers, and non-teaching staff members. The implementation adheres to object-oriented programming principles, ensuring a modular and easily extensible design.

# Functional Requirements

* **Abstract Person Class**

Define an abstract class named Person with shared fields such as name, ID, email, address, and phone number. Include an abstract method named categorize() to define a person's category (teacher or staff).

* **Concrete Classes for Teacher and Staff:**

a. Extend the Person class with a concrete class named Teacher. The Teacher class should have instance variables for specialty, degree (PhD, Master, or Bachelor), and additional fields for its subclasses (part-time or full-time teachers).

b. Extend the Person class with a concrete class named Staff. The Staff class should have instance variables for duty and workload (weekly working hours).

c. Override the toString() and equals() methods for both the Teacher and Staff classes.

* **Specialized Classes for Full-Time and Part-Time Teachers:**

a. Create a FullTimeTeacher class that extends the Teacher class and contains fields specific to full-time teachers.

b. Create a PartTimeTeacher class that extends the Teacher class and contains fields specific to part-time teachers, such as hoursWorked.

* **PayRoll Interface Implementation:**

a. Implement an interface named PayRoll that contains a method named ComputePayRoll().

b. Implement the ComputePayRoll() method in the Teacher and Staff classes with the following specifications:

For full-time teachers, compute the salary as (32 \* degreeRate \* 2) \* 0.85, where degreeRate is 112, 82, or 42 for PhD, Master, and Bachelor, respectively.

For part-time teachers, compute the salary as (hoursWorked \* degreeRate \* 2) \* 0.76, where degreeRate is 112, 82, or 42 for PhD, Master, and Bachelor, respectively.

For staff members, compute the salary as (workload \* 32 \* 2) \* 0.75, with the workload being the weekly working hours, not exceeding 40.

* **Data Loading and Reading:**

During the first execution of the application, load and read teacher and staff data from a text file. This file should contain information on department assignments, and should be updated whenever new teachers or staff are added.

* **GUI Application for Adding and Assigning New Teachers and Staff:**

Create a user-friendly GUI application that allows users to add and assign new teachers and staff members to departments.

* **Department Class Implementation:**

Implement a Department class with the following features:

a. A list of teachers and staff members.

b. A department ID.

c. A dean who is a teacher in the department. If the dean is not a teacher in the department, an exception should be thrown.

* **Data Storage and Updating:**

Store new teachers and staff members added to a department in the text file before exiting the application, generating a new version of the file. Ensure that new teachers and staff are appended to the file if it already exists.

* **Custom Exception Handling:**

Implement custom exception handling for the following cases:

a. Attempting to add a teacher or staff member to an inexistent department (based on department ID).

b. Attempting to add a teacher or staff member that already exists (based on the ID) to the department.

c. Assigning a non-teacher or a teacher from another department as a dean.

* **Test Sets Generation:**

Generate test sets to evaluate the effectiveness of each method in the model, ensuring that the system performs accurately and efficiently.

# Non-Functional Requirements

* **Data Integrity and Validity:**

The system must ensure data integrity and validity through custom exception handling, preventing inconsistencies and errors in the data. This includes checks for duplicate IDs, nonexistent departments, and improper dean assignments.

* **Modularity and Extensibility:**

The software system should be designed using object-oriented programming principles to maintain a modular and easily extensible structure. This allows for future additions or modifications to the system without significant rework or redesign.

* **User-friendly GUI:**

Provide a user-friendly graphical user interface (GUI) for the application, making it easy for users to add and assign new teachers and staff members to departments. The GUI should be intuitive and require minimal training for users to efficiently navigate and perform tasks.

* **Efficient File Handling:**

Implement efficient file handling techniques to ensure quick and reliable data persistence and updates. The system should minimize the time required for reading, writing, and updating the text files containing employee and department data.

* **Scalability:**

The system must be designed to handle a large number of employees and departments without compromising performance. The software should be capable of scaling up to accommodate growth in the university's staff and organizational structure.

* **Maintainability:**

The system should be easy to maintain and update to accommodate future changes in requirements or technologies. This includes adhering to best programming practices, providing clear documentation, and organizing the code in a logical and understandable manner.

* **Performance Optimization:**

Optimize the performance of the system to minimize the time taken for payroll calculations, data retrieval, and other operations. This includes using efficient algorithms and data structures, as well as minimizing the use of resources such as memory and processing power.

* **Security and Privacy:**

Ensure the system adheres to security best practices to protect sensitive employee and department data. Implement access controls, data encryption, and secure file storage techniques to safeguard information and prevent unauthorized access or data breaches.

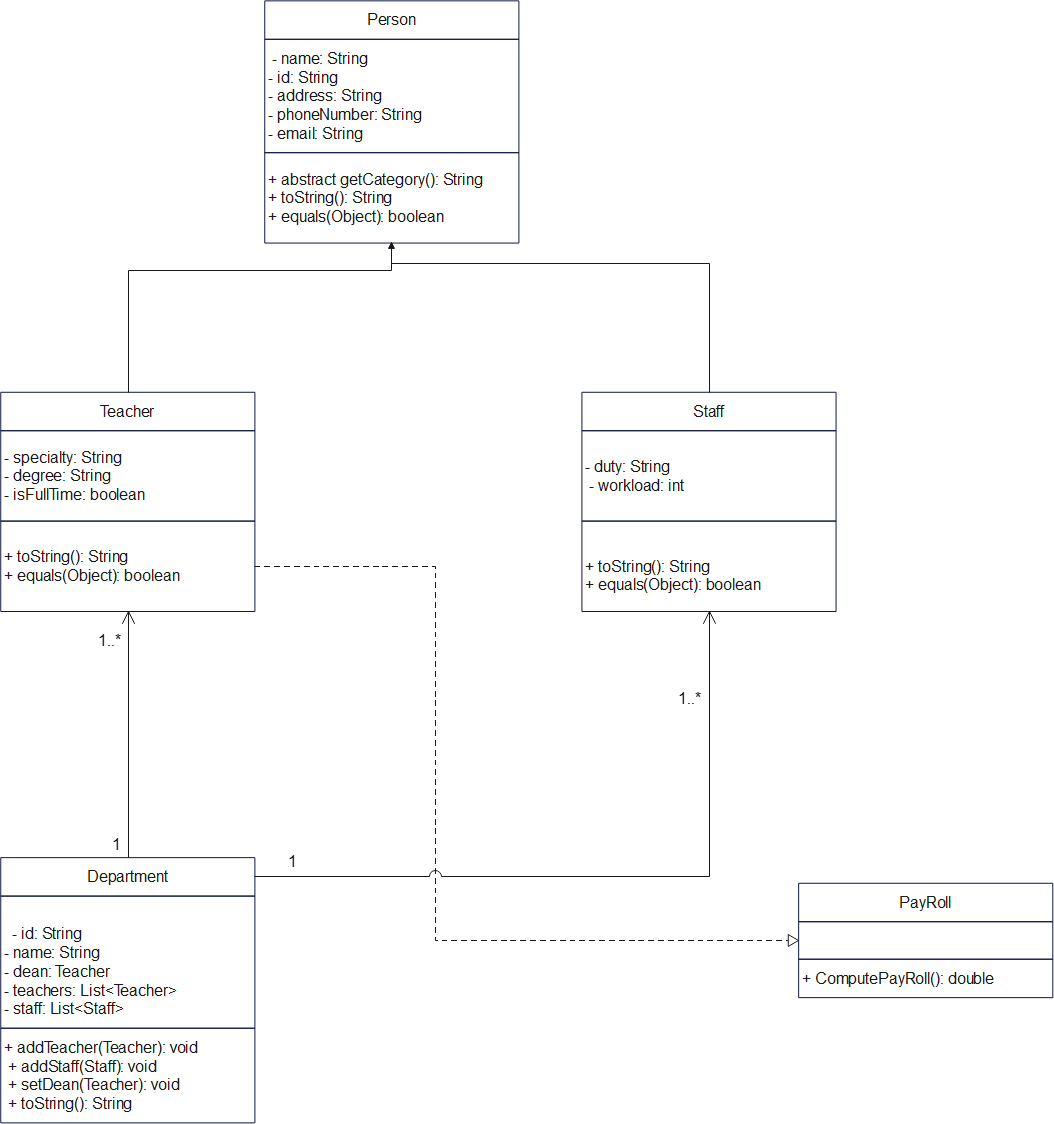
* **Reliability and Robustness:**

The system should be designed to be reliable and robust, minimizing the occurrence of errors, crashes, or unexpected behavior. This includes thorough testing and validation, as well as implementing error handling and recovery mechanisms to ensure the system remains stable and functional.

* **Platform Compatibility:**

The software system should be designed to be compatible with various platforms and operating systems. This allows for greater flexibility in deployment and ensures the system can be used across different environments without major modification.

# UML Class Diagram

****

**Description**

In the given class diagram, there are several relationships between the classes. Here is a description of those relationships:

Inheritance (Generalization):

The Teacher and Staff classes inherit from the Person class. This means they are specialized types of Person and share common attributes and methods. The arrow from the Teacher and Staff classes pointing towards the Person class signifies the inheritance relationship.

Association:

The Department class has an association relationship with the Teacher class through the "dean" attribute (a department has a dean who is a teacher) and the "teachers" attribute (a department has a list of teachers).

The Department class also has an association relationship with the Staff class through the "staff" attribute (a department has a list of staff).These relationships are represented by the diamond arrows pointing from the Department class to the Teacher and Staff classes.

Realization (Implementation):

The Teacher class realizes (implements) the PayRoll interface, meaning it provides an implementation for the ComputePayRoll() method specified in the interface. The dotted arrow from the Teacher class pointing towards the PayRoll interface signifies this realization relationship.

In summary, the main relationships in this diagram are inheritance between Person and its subclasses (Teacher and Staff), association between Department and its related classes (Teacher and Staff), and realization between Teacher and the PayRoll interface.

# Pseudo Code:

1. ComputePayRoll()

function computePayRoll():

declare degreeRate as double

switch(getDegree()):

case "PhD":

degreeRate = 112

break

case "Master":

degreeRate = 82

break

default:

degreeRate = 42

return (32 \* degreeRate \* 2) \* 0.85

1. **ParttimeTeacher()**

**class PartTimeTeacher extends Teacher implements PayRoll, Serializable:**

**declare hoursWorked as int**

**function PartTimeTeacher(id, name, address, phone, email, specialty, degree, hoursWorked):**

**call super(id, name, address, phone, email, specialty, degree)**

**this.hoursWorked = hoursWorked**

**function getHoursWorked():**

**return hoursWorked**

**function setHoursWorked(hoursWorked):**

**this.hoursWorked = hoursWorked**

**function toString():**

**return "PartTimeTeacher{" +**

**"id=" + getId() +**

**", name='" + getName() + '\'' +**

**", specialty='" + getSpecialty() + '\'' +**

**", degree='" + getDegree() + '\'' +**

**", hoursWorked=" + hoursWorked +**

**'}'**

**function equals(o):**

**if (this == o) return true**

**if (!(o instanceof PartTimeTeacher)) return false**

**if (!super.equals(o)) return false**

**PartTimeTeacher that = (PartTimeTeacher) o**

**return hoursWorked == that.hoursWorked**

**function computePayRoll():**

**declare degreeRate as double**

**switch(getDegree()):**

**case "PhD":**

**degreeRate = 112**

**break**

**case "Master":**

**degreeRate = 82**

**break**

**default:**

**degreeRate = 42**

**return (hoursWorked \* degreeRate \* 2) \* 0.76**