/\*

Java Class Project Design

- Java Classes

- UML Class Diagram

- Possible Test Cases

- Once implemented, use for draft sequence diagram

\*/

Based on your project description and the provided code samples, I'll outline a class design that meets the requirements for Company Z's software system. This design will include Java classes for handling employee data and operations as described in your project.

Java Classes

1. GeneralEmployee (Superclass)

- Attributes:

- employeeId (String)

- firstName (String)

- lastName (String)

- email (String)

- ssn (String)

- gender (String)

- pronouns (String)

- race (String)

- dob (Date)

- address (Address)

- Methods:

- Constructors, getters, and setters

- abstract calculatePay(): double

2. FullTimeEmployee (extends GeneralEmployee)

- Additional Attributes:

- baseSalary (double)

- healthCareDeduction (double)

- retirement401k (double)

- Methods:

- calculatePay(): double (overrides superclass method)

3. PartTimeEmployee (extends GeneralEmployee)

- Additional Attributes:

- hourlyWage (double)

- hoursWorked (int)

- Methods:

- calculatePay(): double (overrides superclass method)

4. EmployeeManager

- Attributes:

- employees (ArrayList<GeneralEmployee>)

- Methods:

- addEmployee(GeneralEmployee employee): void

- updateEmployeeData(String employeeId, GeneralEmployee updatedData): void

- searchEmployee(String lastName, String ssnOrEmployeeId): GeneralEmployee

- updatePayRate(String employeeId, double percentageIncrease): void

5. Address (for storing employee address)

- Attributes:

- streetName (String)

- streetNumber (String)

- apartment (String)

- city (String)

- state (String)

- zip (String)

- Methods:

- Constructors, getters, and setters

6. DatabaseManager (For SQL operations)

- Methods:

* connectToDatabase(): Connection
* fetchEmployeeData(): ArrayList<GeneralEmployee>
* updateEmployeeInDatabase(GeneralEmployee employee): void
* ...

UML Class Diagram

- GeneralEmployee:

* -employeeId: String
* -firstName: String
* -lastName: String
* ...
* +calculatePay(): double

- FullTimeEmployee --|> GeneralEmployee

- -baseSalary: double

- -healthCareDeduction: double

- -retirement401k: double

- +calculatePay(): double

- PartTimeEmployee --|> GeneralEmployee

- -hourlyWage: double

- -hoursWorked: int

- +calculatePay(): double

- EmployeeManager

- -employees: ArrayList<GeneralEmployee>

- +addEmployee(employee: GeneralEmployee): void

- ...

- Address

- -streetName: String

- -streetNumber: String

- ...

- DatabaseManager

- +connectToDatabase(): Connection

- +fetchEmployeeData(): ArrayList<GeneralEmployee>

- ...

Test Cases

1. Update Employee Data Test Case

* Scenario: Update the address of a full-time employee.
* Expected: Employee data is updated successfully in the ArrayList and the database.

2. Search Employee Test Case

* Scenario: Search for an employee using last name and SSN.
* Expected: Correct employee is returned if exists.

3. Update Pay Rate Test Case

* Scenario: Give a 4% raise to an employee earning less than a certain amount.
* Expected: Employee's pay rate is updated correctly in the ArrayList and the database.

Sequence Diagrams

\*\*\*FIXME: illustrate the sequence of interactions between objects for operations like increasing salary or hourly wage, adding a new employee, etc., using the classes and their methods outlined above.