Table of Contents

[Full-Time Employee Report 2](#_Toc152606808)

[Part-Time Employee Report 3](#_Toc152606809)

[Database Scema Diagram 4](#_Toc152606810)

[Java Class Diagram 5](#_Toc152606811)

[Test Cases 6](#_Toc152606812)

[Sequence Diagrams 12](#_Toc152606813)

# Full-Time Employee Report

FULL-TIME EMPLOYEE PAYROLL REPORT

Name= Alice Anderson Title=Senior Software Developer alice.anderson@example.com

EMP ID PAY DATE GROSS Federal FedMed FedSS State 401K

1001 2023-09-30 2500.00 750.00 30.00 120.00 240.00 50.00

1001 2023-10-31 2500.00 750.00 30.00 120.00 240.00 50.00

Name= Bob Brown Title=Project Manager bob.brown@example.com

EMP ID PAY DATE GROSS Federal FedMed FedSS State 401K

1002 2023-09-30 3200.00 960.00 38.40 153.60 307.20 64.00

1002 2023-10-31 3200.00 960.00 38.40 153.60 307.20 64.00

Name= Carol Clark Title=IT Specialist carol.clark@example.com

EMP ID PAY DATE GROSS Federal FedMed FedSS State 401K

1003 2023-09-30 2800.00 840.00 33.60 134.40 268.80 56.00

1003 2023-10-31 2800.00 840.00 33.60 134.40 268.80 56.00

Name= David Davis Title=Database Administrator david.davis@example.com

EMP ID PAY DATE GROSS Federal FedMed FedSS State 401K

1004 2023-09-30 2200.00 660.00 26.40 105.60 211.20 44.00

1004 2023-10-31 2200.00 660.00 26.40 105.60 211.20 44.00

Name= Eve Evans Title=Chief Technology Officer eve.evans@example.com

EMP ID PAY DATE GROSS Federal FedMed FedSS State 401K

1005 2023-09-30 5000.00 1500.00 60.00 240.00 480.00 100.00

1005 2023-10-31 5000.00 1500.00 60.00 240.00 480.00 100.00

# Part-Time Employee Report

PART-TIME EMPLOYEE PAYROLL REPORT

Name= Alex Smith Title=Customer Service Rep alex.smith@example.com

EMP ID PAY DATE HOURS RATE GROSS Federal FedMed FedSS State

2001 2023-09-30 80 20.00 1600.00 480.00 19.20 76.80 153.60

2001 2023-10-31 80 20.00 1600.00 480.00 19.20 76.80 153.60

Name= Brittany Jones Title=Web Designer brittany.jones@example.com

EMP ID PAY DATE HOURS RATE GROSS Federal FedMed FedSS State

2002 2023-09-30 60 25.00 1500.00 450.00 18.00 72.00 144.00

2002 2023-10-31 60 25.00 1500.00 450.00 18.00 72.00 144.00

Name= Charles Green Title=Graphic Artist charles.green@example.com

EMP ID PAY DATE HOURS RATE GROSS Federal FedMed FedSS State

2003 2023-09-30 90 18.00 1620.00 486.00 19.44 77.76 155.52

2003 2023-10-31 90 18.00 1620.00 486.00 19.44 77.76 155.52

Name= Dana White Title=Junior Developer dana.white@example.com

EMP ID PAY DATE HOURS RATE GROSS Federal FedMed FedSS State

2004 2023-09-30 85 22.00 1870.00 561.00 22.44 89.76 179.52

2004 2023-10-31 85 22.00 1870.00 561.00 22.44 89.76 179.52

Name= Elliot King Title=Sales Consultant elliot.king@example.com

EMP ID PAY DATE HOURS RATE GROSS Federal FedMed FedSS State

2005 2023-09-30 70 15.00 1050.00 315.00 12.60 50.40 100.80

2005 2023-10-31 70 15.00 1050.00 315.00 12.60 50.40 100.80

# Database Scema Diagram

A computer screen shot of a diagram

Description automatically generated

A white rectangular object with a black line

Description automatically generated with medium confidence

# Java Class Diagram

A screenshot of a computer

Description automatically generated

# Test Cases

**Test Case ID #: 1.1**  
Description: Check if employee's first and last name can be updated correctly  
Test Inputs: First name ‘David’ and last name ‘Mark’  
Expected Results: Employee’s first and last name will be updated to David and Mark respectively.  
Initialization: Instantiate an employee object with employee attributes including first name and last name.  
Dependencies: none  
Test Steps:  
  1.1.1 call the setFirstName(“David”) method on the employee instance.  
  1.1.2 call the setLastName(“Mark”) method on the employee instance.  
 1.1.3 Retrieve the updated first and last name by calling the getFirstName() and getLastName() method on the employee object.

Actual Result:The getFirstName() and getLastName() method return the first name and last name of the employee object as David and Mark respectively.  
Pass/Fail: Pass if the methods getFirstName() and getLastName() return the values David and Mark respectively. Fail otherwise.  
Tear down: Set the employee instance to null.  
  
**Test Case ID #: 1.2**  
Description: Check if employee’s email can be updated correctly  
Test Inputs:  Email = employee@gmail.com  
Expected Results: employee’s email is updated to employee@gmail.com  
Initialization: Instantiate an employee object with employee attributes including email address.  
Dependencies: none.  
Test Steps:  
            1.2.1 call the setEmail(“[employee@gmail.com](mailto:employee@gmail.com)”) method  on the employee instance.  
 1.2.2 Retrieve the updated email by calling the getEmail() method on the employee object.

Actual Result: The getEmail() method returns the Employee’s updated email as employee@gmail.com .  
Pass/Fail: Pass if the getEmail() returns [employee@gmail.com](mailto:employee@gmail.com). Fail otherwise.  
Tear down: Delete the employee instance.  
  
**Test Case ID #: 1.3**

Description: Check if the employee’s address can be updated correctly.

Test Inputs:  StreetName: 1234 Brockett Rd, City: Tucker, State: GA, ZIP: 30033

Expected Results: Employee’s address will be updated to 1234 Brockett Rd, Tucker, GA, 30033.

Initialization: Create an instance of both the employee and address object and initialize them to hold some data. Create an additional instance of the address class to hold the new address.

Dependency: none

Test Steps:  
    1.3.1 call the setAddress(newAddress) on the employee object.  
 1.3.2 Retrieve the updated address by calling the getAddress() method on the employee object.

Actual Result: The getAddress() method returns the updated Employee’s address as 1234 Brockett Rd, Tucker, GA, 30033.

Pass/Fail: Pass if the getAddress() returns the value 1234 Brockett Rd, Tucker, GA, 30033. Fail otherwise.

Tear down: Set the employee and address instances to null.

**Test Case ID #: 1.4**

Description: Check that the payroll information for both full-time and part-time can be updated correctly.

Test Inputs: Input for Full-time payroll information Salary = 60000 , 401K = 0.1, HealthCare = 0.2  
                  Input for Part-time payroll information Wage = 15

Expected Results: The payroll information stored in full-time and part-time employees will be updated correctly.

Initialization: Instantiate a FulltimeEmployee and Part\_Time\_Employee objects with initial payroll information attributes.

Dependency: none

Test Steps for FullTimeEmployee:  
    1.4.1  set the salary of the FulltimeEmployee object using setSalary(60,000)  
    1.4.2  set the 401K of the FulltimeEmployee object using set401K(0.1)  
    1.4.3 set the healthCare of the FulltimeEmployee object using setHealthCare(0.2)  
  
Test Steps for PartTimeEmployee:  
    1.4.1 set the wage of the Part\_Time\_Employee object using setWage(15)  
 1.4.2 Retrieve the updated wage by calling the getWage() method on the employee object.

Actual Result: The getSalary(), get401K(), and getHealthCare() methods returns the updated values 60,000 , 0.1, and 0.2 for the salary, 401K, and healthCare of the FulltimeEmployee object and the getWage() method returns 15 for the wage of the Part\_Time\_Employee object.

Pass/Fail: Pass if the getSalary(), get401K(), and getHealthCare() methods return the values 60,000 , 0.1, and 0.2 respectively for the Fulltime Employee object and getWage() returns 15 for the PartTimeEmployee object. Fail otherwise.

Tear down: Set the FulltimeEmployee and Part\_Time\_Employee object to null.  
  
**Test Case ID# 1.5**  
Description: Check that an employee’s job title can be updated correctly within the employee’s job titles list.  
Test input:  Assign employee a job title called ‘ Mobile App Developer’  
Expected Results: The getJobTitle() will return the  updated job title  ‘Mobile App Developer’.  
  
Initialization: Instantiate an Employee object. Instantiate the EmployeeJobTitle  with a single job title and add it to the employeeJobTitle list attribute of the Employee object.  
  
Test Steps  
    1.5.1  Select the EmployeeJobTitle object that needs to be updated within the EmployeeJobTitles list.  
    1.5.2  Call the setJobTitle(‘Mobile App Developer’) on the EmployeeJobTitle object to update the job title of the employee object.  
      1.5.3 Retrieve the updated job title from the EmployeeJobTitle list using the getJobTitle().   
  
Actual Result: the getJobTitle() method returns Mobile App Developer as the new job title of the Employee object.  
Pass/Fail criteria: Pass if the getJobTitle() returns Mobile App Developer for the employee’s job title, fail otherwise.  
    Tear Down: Set the Employee and EmployeeJobTitle object to null.  
  
**Test Case ID #: 1.6**

Description: Check if the employee’s demographic data can be updated correctly.

Test Inputs:  SSN: 111-22-3333 DOB: ‘01-02-1990’ Gender: ‘Male’ Race: ‘African American’

Expected Results: Employee’s demographic data will be updated to SSN: 111-22-3333 DOB: ‘01-02-1990’ Gender: ‘Male’ Race: ‘African American’

Initialization: Create an instance of both the employee and a demographic object and initialize them to hold some data. Create an additional instance of the new demographic object to hold the new demographic data.

Dependency: none

Test Steps:  
    1.6.1 call the setDemography(newDemography) on the employee object.  
 1.6.2 retrieve the updated Demography by calling the getDemography() method on the employee object.

Actual Result: The getDemography() method returns the updated Employee’s demographic data as SSN: 111-22-3333 DOB: ‘01-02-1990’ Gender: ‘Male’ Race: ‘African American’.

Pass/Fail: Pass if the getDemography() returns the value SSN: 111-22-3333 DOB: ‘01-02-1990’ Gender: ‘Male’ Race: ‘African American’. Fail otherwise.

Tear down: Set the employee and Demographics instances to null.

**Test Case 2.1 (B) - Search for Employee**

1.     **Test Case ID**: 2.1

2.     **Test Case Description**: Verify that the employee search functionality correctly retrieves an employee's record when searched by last name in combination with either SSN or employee ID.

3.     **Test Inputs**: Various combinations of input will be used to test various scenarios. The parameters used when executing test case are:

·       Employee’s last name, i.e. “Doe”

·       Employee ID, i.e. “EMP001”

·       Employee’s SSN, i.e. “123-45-6789”

4.     **Expected Results**:

·       When the last name and SSN are used as inputs, the system returns the correct employee record matching both.

·       When the last name and Employee ID are used as inputs, the system returns the correct employee record matching both.

5.     **Initialization**:

·       Ensure the testing environment is set up with the employee management application running.

·       Verify that the database is populated with test data, including an employee with the last name "Doe", SSN "123-45-6789", and Employee ID "EMP001".

6.     **Dependency**:

·       The system must be able to connect to the database without issues.

·       The user conducting the test must have the necessary permissions to access the search functionality.

7.     **Test Steps**:

**Step 1**: Log in to the system.

**Step 2**: Navigate to the employee search interface

**Step 3**: Enter last name of known employee “Doe” and SSN “123-45-6789” in the respective fields and initiate search.

**Step 4**: Clear all fields and repeat with last name “Doe” and Employee ID “EMP001” into respective fields.

8.     **Postconditions**: System remains operational and no changes were made to the employee’s records as a result of the operation.

9.     **Test Pass/Fail Criteria**: The test case passes if all steps execute successfully and the data returned by the system matches the expected data for the employee record. Otherwise it fails.

10. **Tear Down**: Log out of system. Close any open connections to DB.

**Actual Result (Example)**:

|  |  |  |
| --- | --- | --- |
| Step 1: |  | Successfully logged into the system. |
| Step 2: |  | Searched for the employee by last name and SSN and successfully found the employee “Doe” with SSN “123-45-6789” |
| Step 3: |  | Searched for the same employee by last name but using employee ID instead and successfully found the employee “Doe” with Employee ID "EMP001" |

**Test Case 2.2 (B) - Search for Non-Existent Employee**

1.     **Test Case ID**: 2.2

2.     **Test Case Description**: Ensure that the employee search functionality does not retrieve any record when searched using details of a non-existent employee.

3.     **Test Inputs**: Various combinations of input will be used to test various scenarios. The parameters used when executing test case are:

·       Employee’s last name, i.e. “NonExistentLastName”

·       Employee ID, i.e. “NONEMP999”

·       Employee’s SSN, i.e. “999-99-9999”

4.     **Expected Results**:

·       When a last name and SSN that do not match any employee record are used as inputs, the system should not return any record.

·       When a last name and Employee ID that do not match any employee record are used as inputs, the system should not return any record

5.     **Initialization**:

·       Ensure the testing environment is set up with the employee management application running.

·       Confirm that there are no employees in the database with the last name “NonExistentLastName” and the provided SSN and Employee ID.

6.     **Dependency**:

·       The system must be able to connect to the database without issues.

·       The user conducting the test must have the necessary permissions to access the search functionality.

7.     **Test Steps**:

**Step 1**: Log in to the system.

**Step 2**: Navigate to the employee search interface

**Step 3**: Confirm that no employee currently exists with your search criteria

**Step 4**: Enter last name “NonExistentLastName” and SSN “999-99-9999” in the respective fields and initiate search.

**Step 5**: Clear all fields and repeat with last name “NonExistentLastName” and Employee ID “NONEMP999” into respective fields.

8.     **Postconditions**: System remains operational and no new records were created as a result of the search operation.

9.     **Test Pass/Fail Criteria**: The test case passes if the system does not return any employee records for the non-existent employee data. The test fails if any records are returned.

10. **Tear Down**: Log out of system. Close any open connections to DB.

**Actual Result (Example)**:

|  |  |  |
| --- | --- | --- |
| Step 1: |  | Successfully logged into the system. |
| Step 2: |  | No employee was found for last name “NonExistentLastName” and SSN “999-99-9999”. |
| Step 3: |  | No employee was found for last name “NonExistentLastName” and Employee ID “NONEMP999” |

**Test Case #3.1**

**Description:** Update the salary of full-time employees who earn less than a specific amount

**Test Inputs**: New salary amount, newSal = 120000. Salary threshold, maxSal = 80000

**Expected Results:** The respective salary amounts for employees below the threshold will be updated, and those above the threshold will be unchanged.

**Initialization:** Instantiate multiple FulltimeEmployee objects.

**Dependency:** none

**Test Steps:**

3.1.1 set the salary of one FulltimeEmployee object to be below maxSal [setSalary()].

3.1.2 set the salary for another FulltimeEmployee object to exceed maxSal [setSalary()].

3.1.3 use the method call increaseSalaryIfBelowThreshold(maxSal, newSal)

3.1.4 retrieve the updated salaries of both FulltimeEmployee objects using getSalary().

**Actual Result:** The getSalary() method for the FulltimeEmployee object below the threshold returns 120000. The getSalary() method for the FullTimeEmployee above the threshold returns the object's initial, unmodified salary.

**Pass/Fail:** Pass if getSalary() returns an updated value for one employee, and retains the value of the other. Fail if no salaries are changed, or if both salaries are changed.

**Tear down:** Set the FulltimeEmployee instances to null

**Test Case #3.2**

**Description:** Update the hourly wage of part-time employees who earn less than a specific amount

**Test Inputs**: New wage amount, newWage = 16. Wage threshold, maxWage = 13

**Expected Results:** The respective wage amounts for employees below the threshold will be updated, and those above the threshold will be unchanged.

**Initialization:** Instantiate multiple Part\_Time\_Employee objects.

**Dependency:** none

**Test Steps:**

3.1.1 set the wage of one Part\_Time\_Employee object to be below maxWage.

3.1.2 set the wage of another Part\_Time\_Employee object to exceed maxWage.

3.1.3 call the method increaseWageIfBelowThreshold(maxWage, newWage)

3.1.4 retrieve the updated wages of both Part\_Time\_Employee objects using getWage().

**Actual Result:** The getWage() method for the Part\_Time\_Employee object below the threshold returns 16. The getWage() method for the Part\_Time\_Employee above the threshold returns the object’s initial, unmodified wage.

**Pass/Fail:** Pass if getWage() returns an updated value for one employee, and retains the value of the other. Fail if no wages are changed, or if all wages are changed.

**Tear down:** Set the Part\_Time\_Employee instances to null

# Sequence Diagrams

1. **Sequence diagram to increase salary by % if current base is less than a given amount**
2. **Sequence diagram to increase hourly wage by % if current base is less than a given amount**
3. **Sequence diagram to add new employee to employee database**

**A diagram of a company

Description automatically generated**