a)

- The objectives of the test are to determine whether the actual outcome of the program meets the expected outcome results.

- This includes determining the maximum number of users the system can handle.

- Another purpose of the test is to determine how the program behaves under wrong user input and whether it will validate it. An example is when the user inputs a value that's not an integer when asked to enter their work hours.

- Testing the program will involve test cases that make sure that the correct formulas for calculating are correct and produce the correct results based on the user input given.

- Evaluate how the program performs with different volumes of data, including large datasets. Check if the program handles large amounts of data efficiently, and measure the time taken for calculations and data processing. Ensure that the program does not experience performance issues or slowdowns with increased load.

- The maximum amount of users is unlimited since it applies to all the workers within a factory.

b)

Overview

- The program aims to make sure that the program works correctly with new data inputs and existing data that has been previously processed and stored in the system.

Objectives

- Verify the program handles new data inputs correctly.
- Ensure the program processes existing data without errors or crashes.
- Validate the correctness of pay calculations.
- Confirm the program's ability to read from and write to the data file.
- Check user input handling and error messaging.
- Test retirement plan functionality.

Test Phases

Cycle 1: New Data Testing

- Objective: Ensure the program works correctly with fresh data inputs.
- Test Cases:
    - Basic Input Handling:
    - Test inputting valid hours worked and shift number.
    - Test inputting invalid hours worked and shift numbers (e.g., negative numbers, non-integer values).

- **Pay Calculation:**
    - Test calculations for different shift numbers without overtime.
    - Test calculations for different shift numbers with overtime.

- **Retirement Plan:**
    - Test shifts 2 and 3 with and without opting into the retirement plan.
    - Verify retirement deductions are correctly applied and displayed.

- **File Operations:**
    - Test writing new hours worked to the file.
    - Ensure the file is created if it does not exist.

Cycle 2: Existing Data Testing

- Ensure the program can handle data that has been previously processed or is stored in the system.

Test Cases:

1. Verify the Program Handles New Data Inputs Correctly

Test Case 1: Valid New Data Input

- Input valid hours worked and a valid shift number.
    1. Run the program.
    2. Enter 40 for hours worked.
    3. Enter 1 for the shift number.
- **Expected Result**: The program accepts the input, calculates pay, and displays the correct details.

Test Case 2: Invalid Hours Worked Input

- Input invalid hours worked (e.g., negative number).
    1. Start the program.
    2. Enter -5 for hours worked.
- **Expected Result**: The program displays an error message indicating invalid input.

Test Case 3: Invalid Shift Number Input

- Input an invalid shift number.
    - Start the program.
    - Enter 40 for hours worked.
    - Enter 4 for the shift number.
- **Expected Result**: The program displays an error message indicating invalid input.

2. Ensure the Program Processes Existing Data Without Errors or Crashes

Test Case 4: Process Existing Valid Data

- Process data from an existing file with valid data.
    1. Ensure hours_worked.txt contains valid data entries.
    2. Start the program.
- **Expected Result**: The program reads the existing data and processes it without errors.

Test Case 5: Process Existing Corrupted Data

- Process data from an existing file with corrupted data.
    1. Corrupt hours_worked.txt file with invalid entries.
    2. Start the program.
- **Expected Result**: The program handles the corrupted data gracefully, displaying appropriate error messages.

3. Confirm the Program's Ability to Read from and Write to the Data File

Test Case 6: Read from Existing File

- Read data from an existing file.
    1. Ensure hours_worked.txt contains valid data entries.
    2. Start the program.
- **Expected Result**: The program reads and displays the data from the file correctly.

Test Case 7: Write to File

- Write new data to the file.
    1. Start the program.
    2. Enter 30 for hours worked.
    3. Enter 1 for the shift number.
- **Expected Result**: The program writes the new data to the file without errors.

4. Check User Input Handling and Error Messaging

Test Case 8: Handle Invalid Input Gracefully

- Enter invalid data and check error handling.
    1. Start the program.
    2. Enter 'ABC' for hours worked.
- **Expected Result**: The program displays an error message indicating invalid input.

5. Test Retirement Plan Functionality

Test Case 9: Say yes to Retirement Plan for Shift 2

- Verify retirement plan calculation for shift 2 with retirement contribution.
    1. Start the program.
    2. Enter 40 for hours worked.
    3. Enter 2 for the shift number.
    4. Say yes to the retirement plan.
- **Expected Result**: The program correctly calculates and displays retirement deduction and net pay.

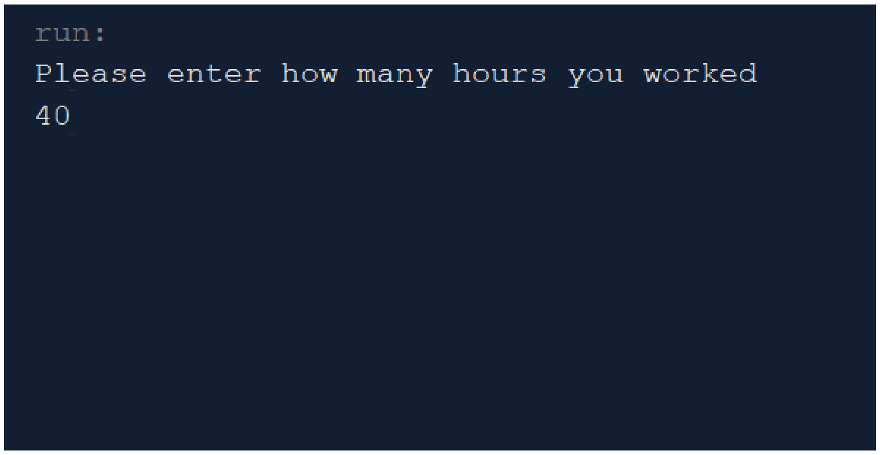
Test Case 10: Say no to Retirement Plan for Shift 2

- Verify pay calculation for shift 2 without retirement contribution.
- **Steps**:
  1. Start the program.
  2. Enter 40 for hours worked.
  3. Enter 2 for the shift number.
  4. Say no to the retirement plan for shift 2.
- **Expected Result**: The program calculates and displays correct regular pay and net pay without retirement deduction.

c)

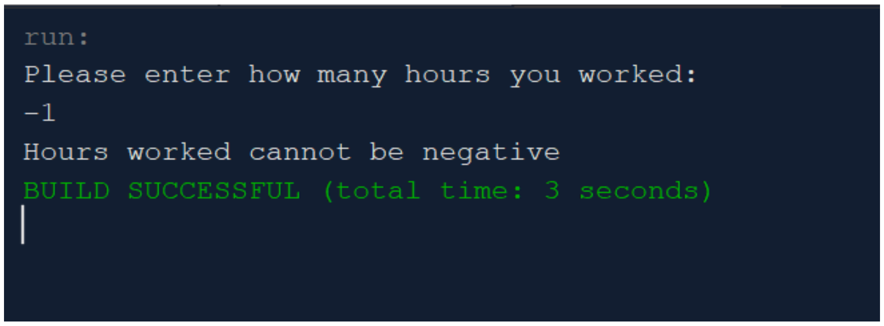
Cycle 1

1. Valid New Input:



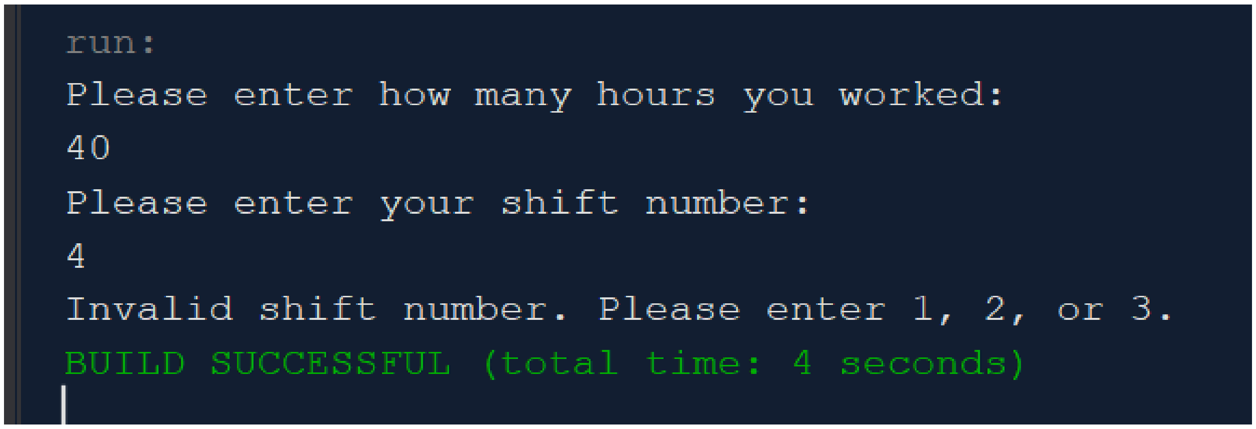
```
run:
Please enter how many hours you worked
40
```

2. Invalid work hours



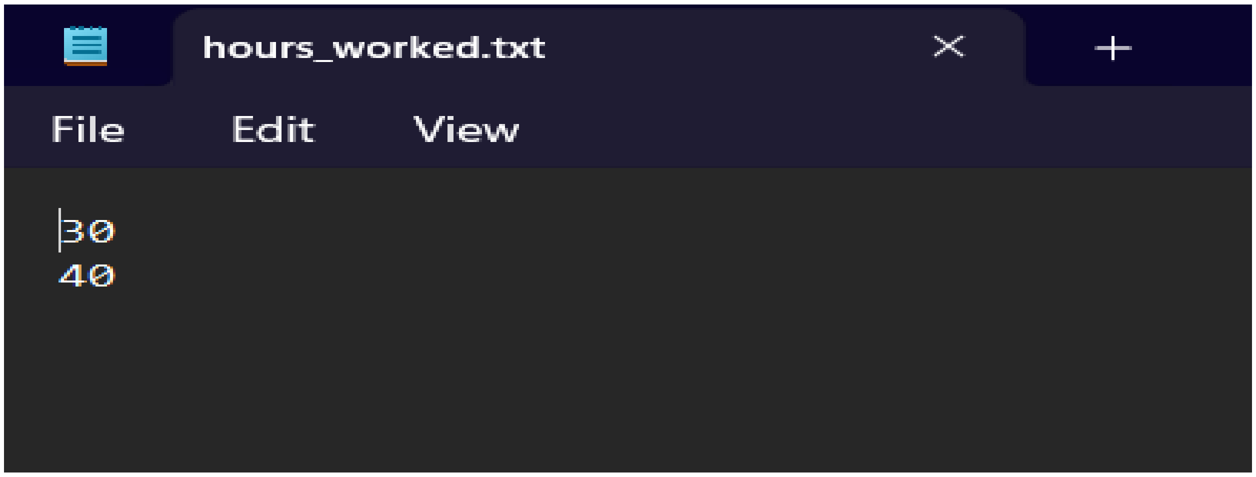
```
run:
Please enter how many hours you worked:
-1
Hours worked cannot be negative
BUILD SUCCESSFUL (total time: 3 seconds)
```

3. Invalid Shift number



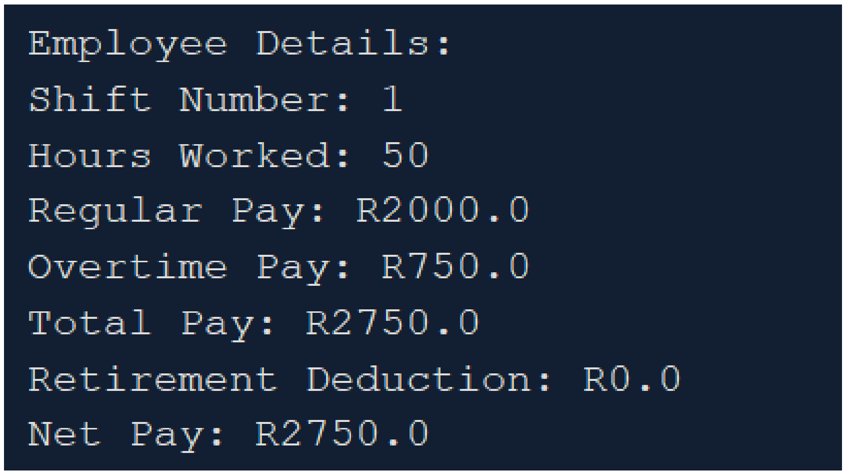
```
run:
Please enter how many hours you worked:
40
Please enter your shift number:
4
Invalid shift number. Please enter 1, 2, or 3.
BUILD SUCCESSFUL (total time: 4 seconds)
```

4. Write to file



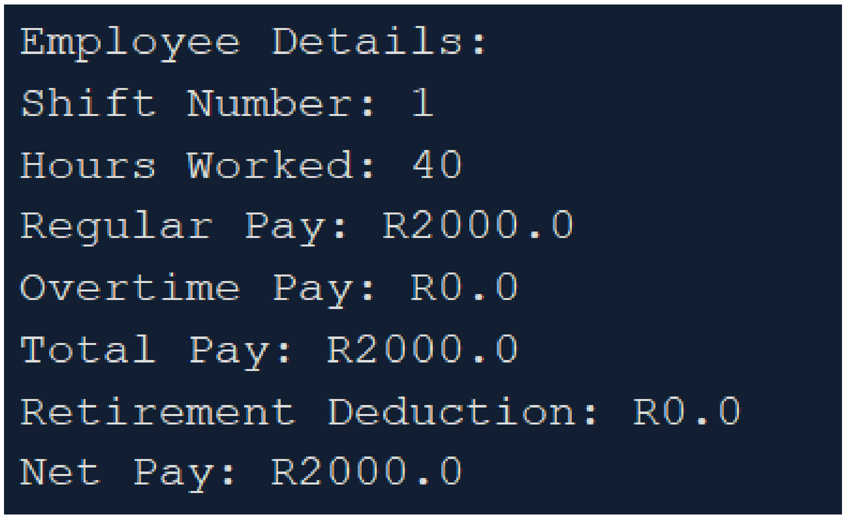
```
30
40
```

5. Calculation with Overtime



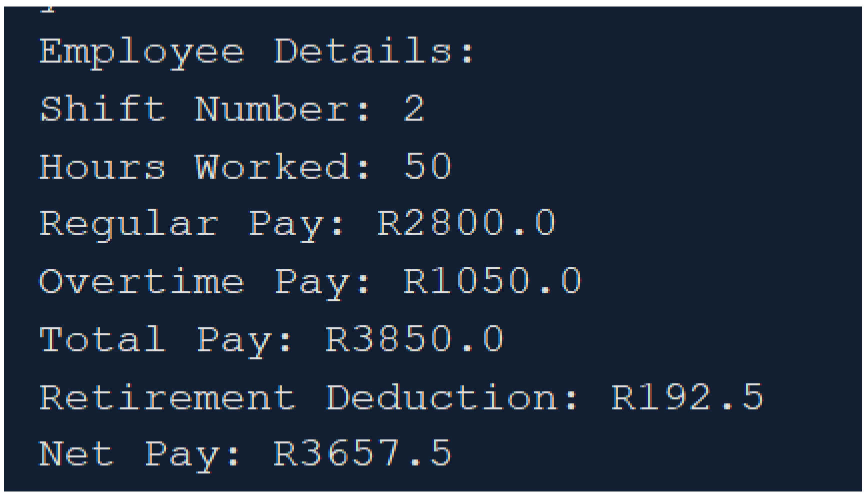
```
Employee Details:
Shift Number: 1
Hours Worked: 50
Regular Pay: R2000.0
Overtime Pay: R750.0
Total Pay: R2750.0
Retirement Deduction: R0.0
Net Pay: R2750.0
```

6. Calculation without Overtime



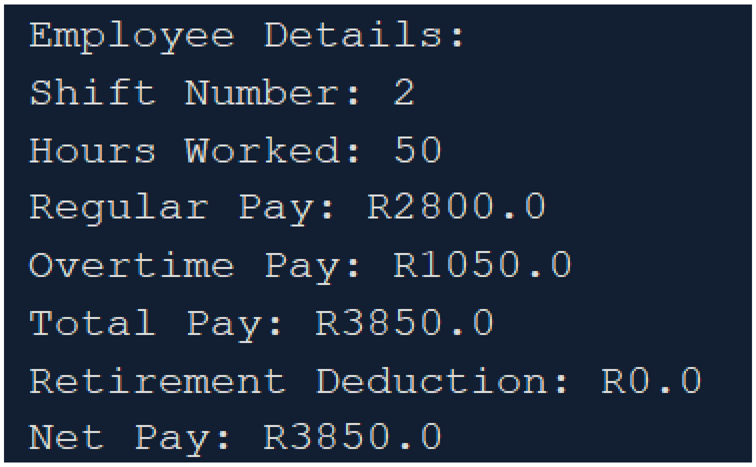
```
Employee Details:
Shift Number: 1
Hours Worked: 40
Regular Pay: R2000.0
Overtime Pay: R0.0
Total Pay: R2000.0
Retirement Deduction: R0.0
Net Pay: R2000.0
```

7. Calculation with retirement



```
Employee Details:
Shift Number: 2
Hours Worked: 50
Regular Pay: R2800.0
Overtime Pay: R1050.0
Total Pay: R3850.0
Retirement Deduction: R192.5
Net Pay: R3657.5
```

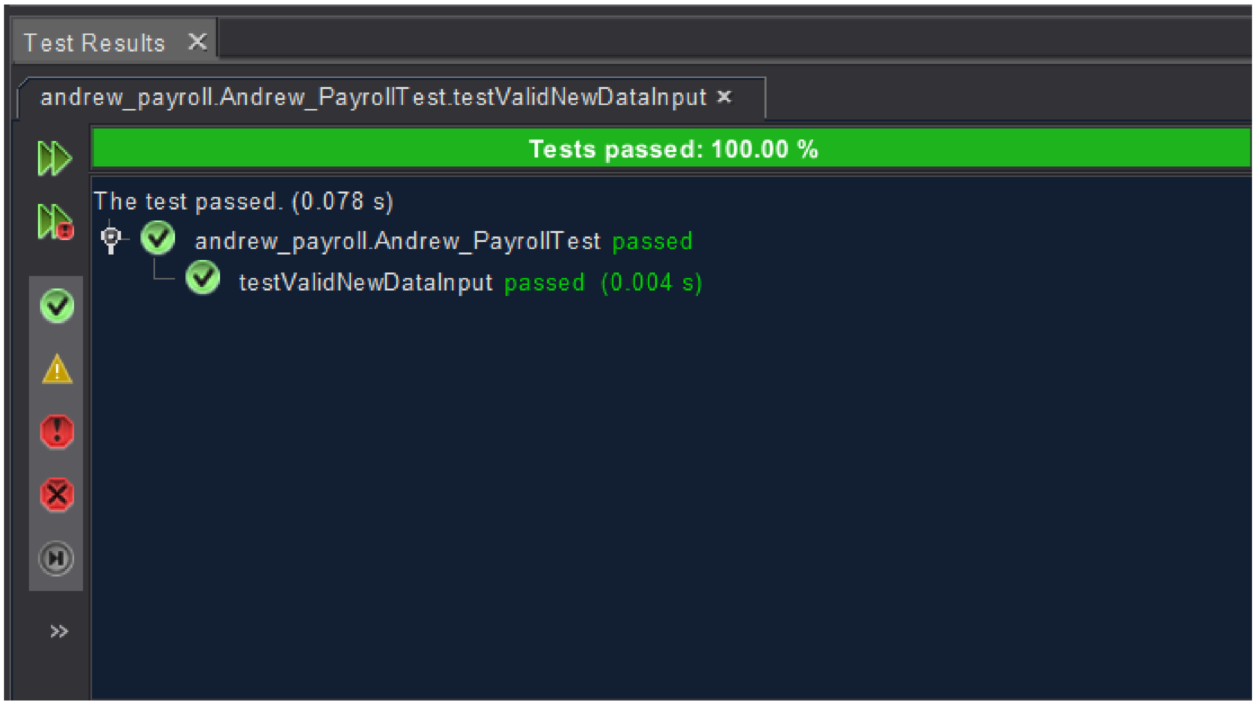
8. Calculation without retirement



```
Employee Details:
Shift Number: 2
Hours Worked: 50
Regular Pay: R2800.0
Overtime Pay: R1050.0
Total Pay: R3850.0
Retirement Deduction: R0.0
Net Pay: R3850.0
```

Cycle 2

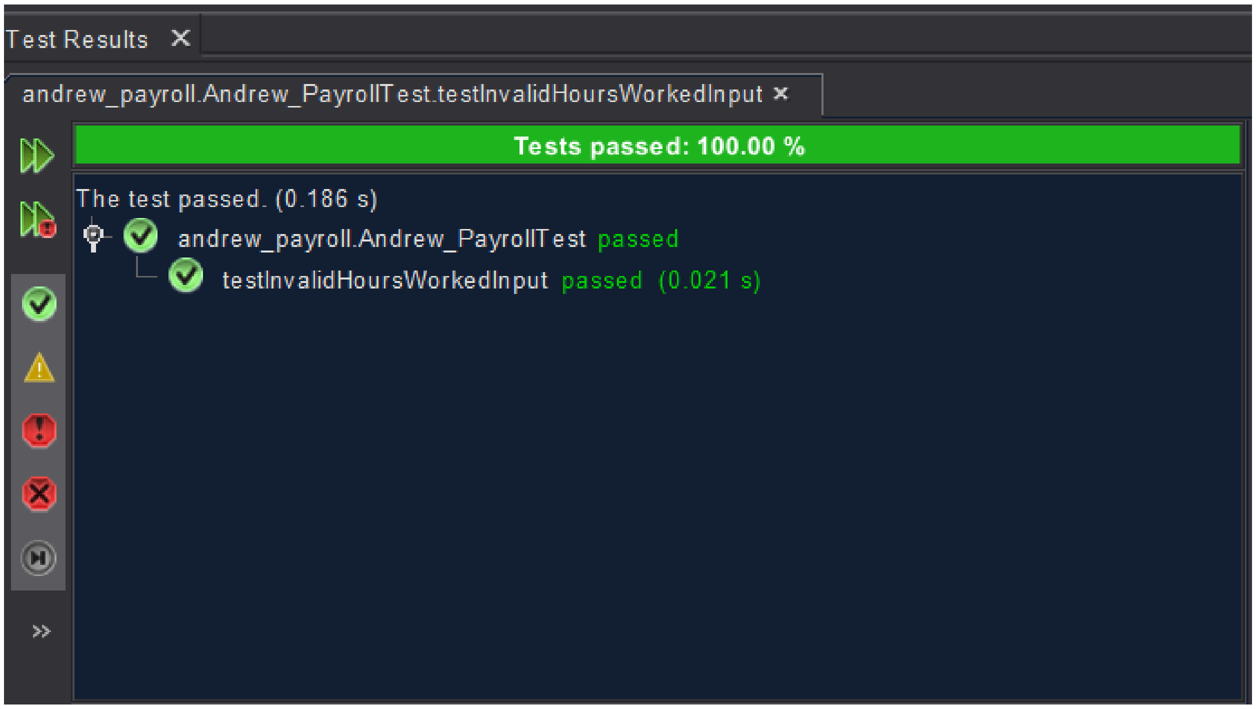
Test Case 1: Valid New Data Input



```
Test Results ✕
andrew_payroll.Andrew_PayrollTest.testValidNewDataInput ×
Tests passed: 100.00 %
The test passed. (0.078 s)
  andrew_payroll.Andrew_PayrollTest passed
    testValidNewDataInput passed (0.004 s)
```

- **Expected Result:**
    - **Regular Pay**: 40×50=R2000
    - **Overtime Pay**: R0 (No overtime)
    - **Total Pay**: R2000
    - **Retirement Deduction**: R0 (No retirement plan)
    - **Net Pay**: R2000
- **Actual Result:**
    - **Regular Pay**: (Check if it matches R2000)
    - **Overtime Pay**: (Check if it matches R0)
    - **Total Pay**: (Check if it matches R2000)
    - **Retirement Deduction**: (Check if it matches R0)
    - **Net Pay**: (Check if it matches R2000)

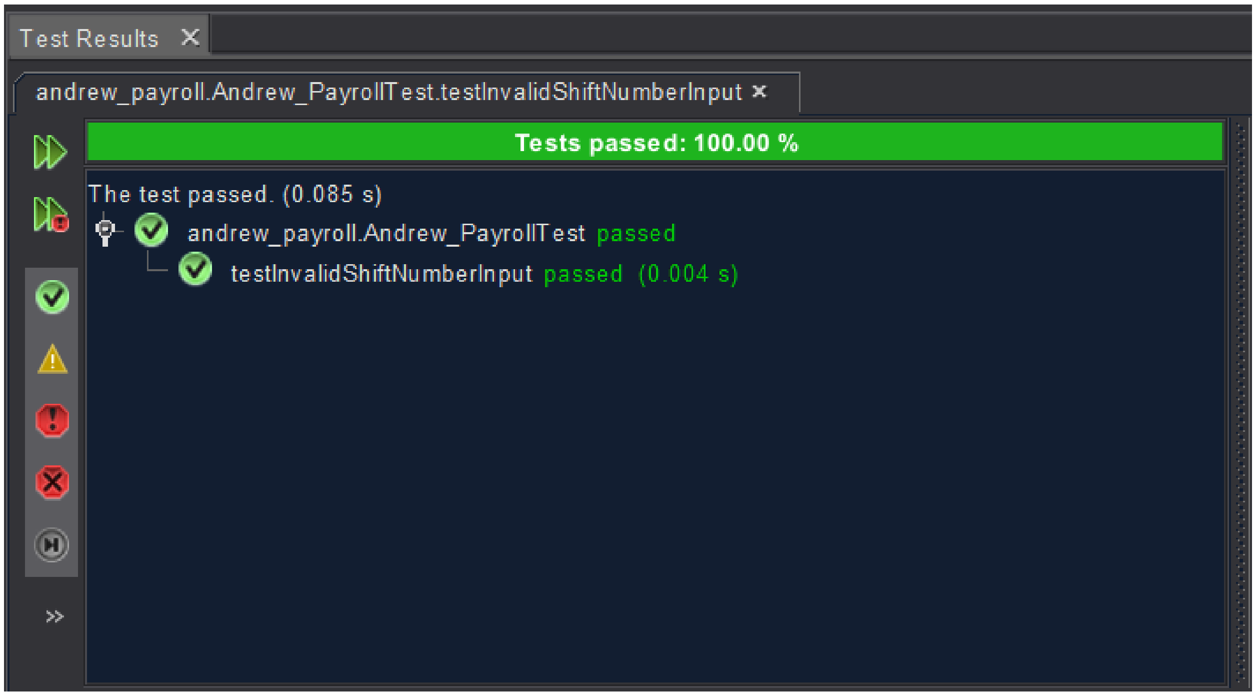
Test Case 2: Invalid Hours Worked Input


Test Results
andrew_payroll.Andrew_PayrollTest.testInvalidHoursWorkedInput

```
Tests passed: 100.00 %
The test passed. (0.186 s)
  andrew_payroll.Andrew_PayrollTest passed
    testInvalidHoursWorkedInput passed (0.021 s)
```

- **Expected Result:**
  - Exception: IllegalArgumentException for negative hours.
- **Actual Result:**
  - Exception: (Check if it is IllegalArgumentException)

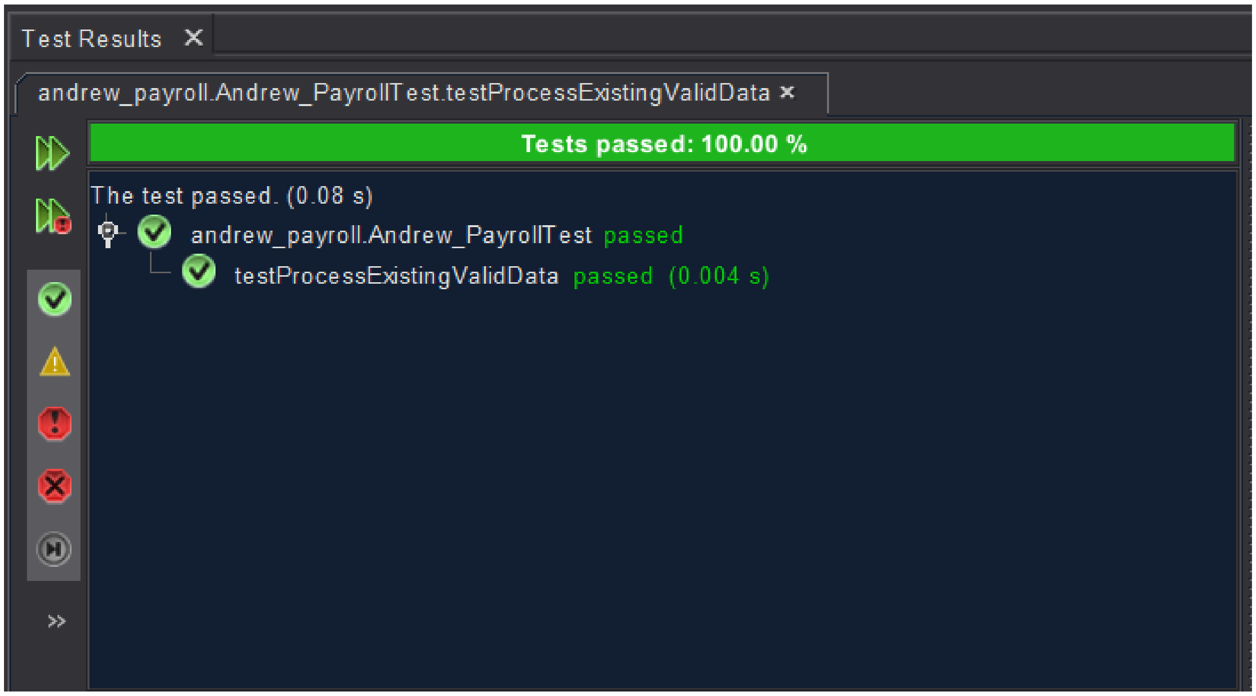
Test Case 3: Invalid Shift Number Input



```
Test Results
andrew_payroll.Andrew_PayrollTest.testInvalidShiftNumberInput
Tests passed: 100.00 %
The test passed. (0.085 s)
andrew_payroll.Andrew_PayrollTest passed
  testInvalidShiftNumberInput passed (0.004 s)
```

- **Expected Result:**
    - **Regular Pay**: 0
    - **Overtime Pay**: 0
    - **Total Pay**: 0
    - **Retirement Deduction**: 0
    - **Net Pay**: 0
- **Actual Result:**
    - **Regular Pay**: (Check if it matches 0)
    - **Overtime Pay**: (Check if it matches 0)
    - **Total Pay**: (Check if it matches 0)
    - **Retirement Deduction**: (Check if it matches 0)
    - **Net Pay**: (Check if it matches 0)

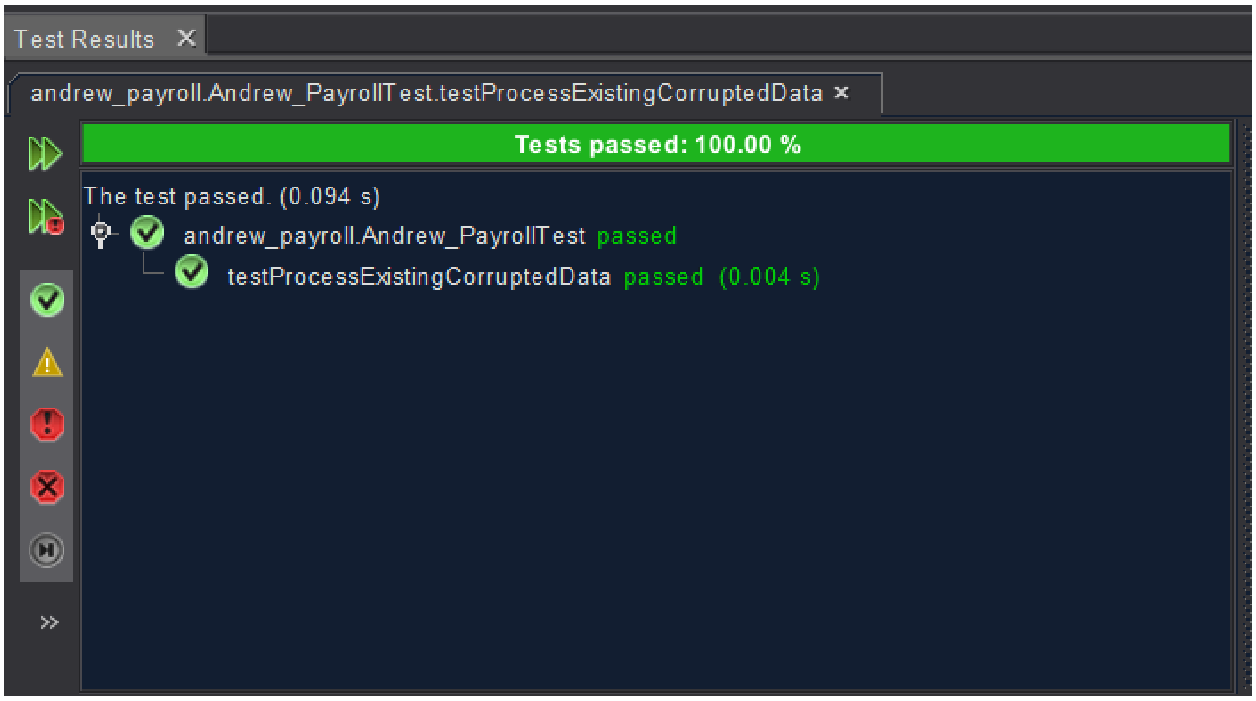
Test Case 4: Process Existing Valid Data


Test Results
andrew_payroll.Andrew_PayrollTest.testProcessExistingValidData

```
Tests passed: 100.00 %
The test passed. (0.08 s)
  andrew_payroll.Andrew_PayrollTest passed
    testProcessExistingValidData passed (0.004 s)
```

- **Expected Result:**
  - The program processes the file data without errors.
- **Actual Result:**
  - (Check if the file is processed without errors)

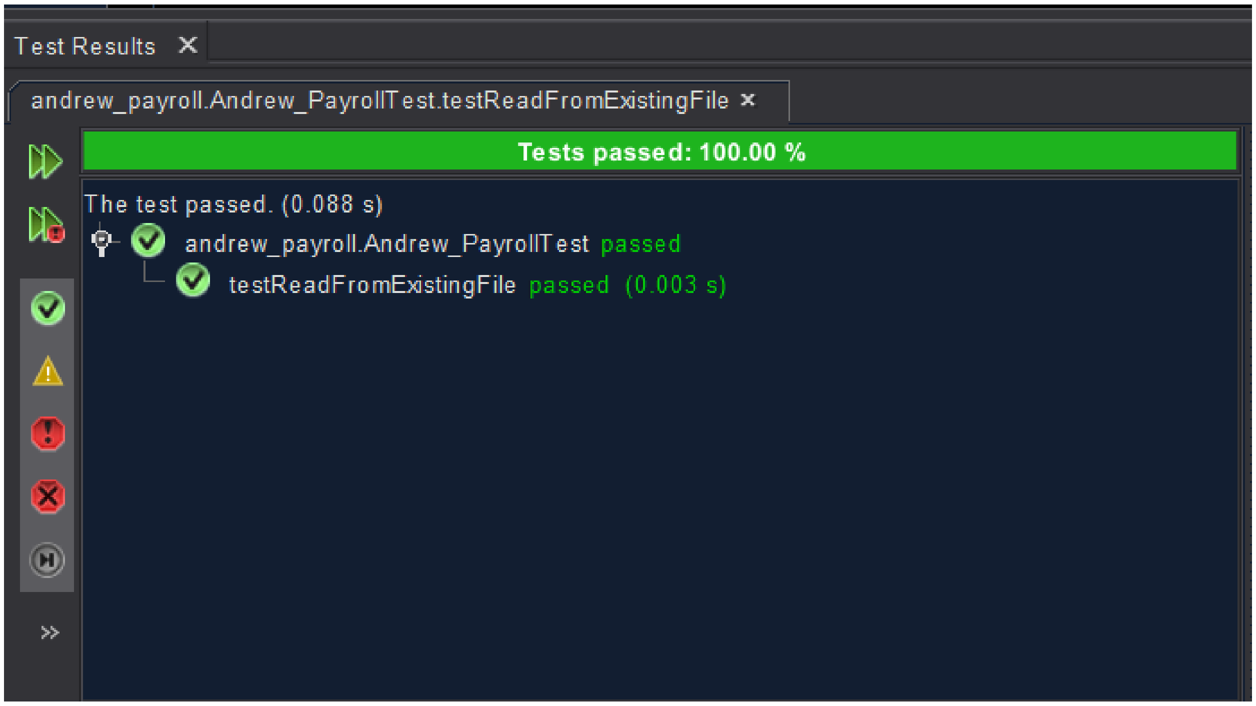
Test Case 5: Process Existing Corrupted Data



```
Test Results
andrew_payroll.Andrew_PayrollTest.testProcessExistingCorruptedData
Tests passed: 100.00 %
The test passed. (0.094 s)
andrew_payroll.Andrew_PayrollTest passed
    testProcessExistingCorruptedData passed (0.004 s)
```

- **Expected Result:**
  - **Exception**: IOException for corrupted data.
- **Actual Result:**
  - **Exception**: (Check if it is IOException)

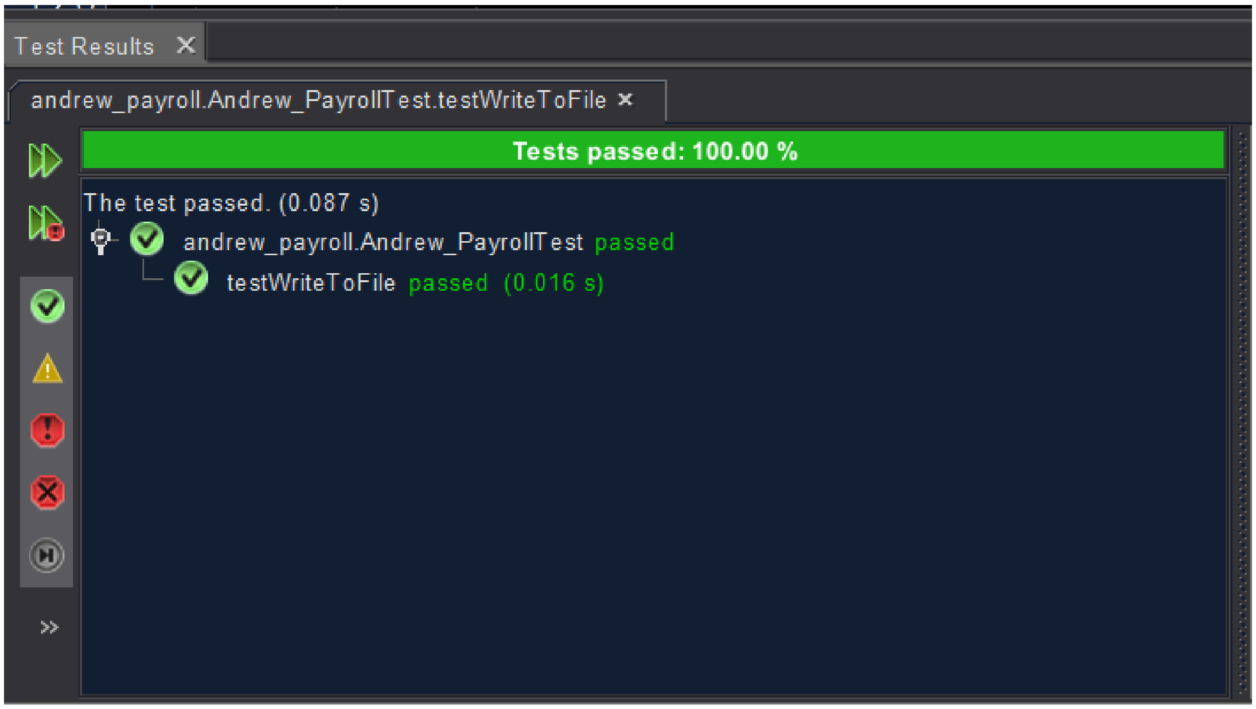
Test Case 6: Read from Existing File



```
Test Results
andrew_payroll.Andrew_PayrollTest.testReadFromExistingFile
Tests passed: 100.00 %
The test passed. (0.088 s)
  andrew_payroll.Andrew_PayrollTest passed
    testReadFromExistingFile passed (0.003 s)
```

- **Expected Result:**
    - The program reads and displays the data correctly.
- **Actual Result:**
    - (Check if the file data is read and displayed correctly)

Test Case 7: Write to File



```
Tests passed: 100.00 %
The test passed. (0.087 s)
andrew_payroll.Andrew_PayrollTest passed
    testWriteToFile passed (0.016 s)
```

- **Expected Result:**
    - The file contains the new entry "30".
- **Actual Result:**
    - (Check if the file contains the entry "30")