Name: William Ewanchuk

Ccid: wewanchu

SID: 1692036

Section: D41

1. Correct Salary Calculation:

- Purpose: Evaluate if the pay method accurately computes the take-home salary given a range of valid inputs. This includes base salary, snacks expenses, and bonus percentages adhering to defined constraints.

- Details: Tests are crafted to cover typical use cases, ensuring that the deduction of snack expenses and the computation of bonuses are accurately reflected in the final pay. Specific focus is given to edge scenarios such as having zero bonuses or a maximum base salary of $1000.

- Example Case: With inputs of a $100 base salary, $50 spent on snacks, and a 10% bonus, the expected pay output should be $55. This confirms that the method correctly calculates deductions and applies the percentage bonus.

2. Illegal Argument Handling:

- Purpose: Ensure that the method robustly handles inputs failing to meet stipulated constraints by throwing an IllegalArgumentException.

- Details: The suite tests scenarios including salaries exceeding $1000, any form of negative input values, and snack expenses surpassing the salary. These tests validate that the method's parameter validations are preventing unrealistic or illegal situations.

- Example Case: Inputting a salary of $1200 ought to trigger an exception as this exceeds the maximum allowable salary according to the requirements.

3. Null Input Handling:

- Purpose: Validate that the method appropriately addresses cases where null values are input by throwing a NullPointerException, thereby preventing unintended behaviors or system crashes.

- Details: Each parameter is separately tested with null to ensure thorough coverage and reliable error messaging whenever null inputs compromise expected operations.

- Example Case: Invoking the method with pay(null, 50.0, 5) is expected to produce a NullPointerException, indicating that the method cannot proceed with inadequate information.

4. Input Type Validation:

- Purpose: Ascertain that only valid data types, precisely Java objects as specified (i.e., Double and Integer), are accepted by the method, rejecting incompatible types through typical parsing failures.

- Details: Tests involve attempting to parse non-numeric strings as salary, snack amounts, and bonuses to verify enforcement of correct data formats and types.

- Example Case: Attempting to parse pay("abc", "50.0", "5") should culminate in a NumberFormatException, indicating a failure in data conversion due to incorrect format.

5. Boundary and Edge Cases:

- Purpose: Confirm that the method reliably executes under boundary conditions, embodying values at the permissible extremes as well as zero-value scenarios.

- Details: Tests are deployed with minimum non-negative values, zero expenses, and maximum bonus percentages to ensure the method's robustness and accuracy extending to its operational envelope.

- Example Case: Providing both salary and snacks as $0 should naturally yield a pay of $0, validating the method's logic in scenarios of no economic exchange.

Tools and Framework:

- JUnit: Employing JUnit allows for high precision in assertions such as assertEquals with a delta of 0.001, governing acceptable floating-point discrepancies. This offers structured and automated testing of method functionality as well as exception handling.

import org.junit.jupiter.api.Test:

- Provides the annotation @Test for standard test methods in JUnit 5.

import org.junit.jupiter.params.ParameterizedTest:

- Enables parameterized tests that allow for running the same test logic with different inputs.

import org.junit.jupiter.params.provider.CsvSource:

- Supplies a set of values in a CSV (Comma-Separated Values) format for parameterized tests.

import org.junit.jupiter.params.provider.ValueSource:

- Supplies a single set of values (like primitives or strings) for iterating over parameterized tests.

Conclusion:

This comprehensive test plan is structured to procedurally verify both the functional accuracy and the reliability of the pay method across a large test suit. It ensures that user constraints are adhered to and that the framework graciously responds to invalid input scenarios, fulfilling the assignment's requirement for an infallible salary computation tool.