Case study for implementing Java application with TDD

The following is the specification of the Employee Pay-roll Application case study that is to be implemented as part of applying TDD in the class room.

The user stories to be implemented are as follows.

1. Employee is working with a company and gets salary credited to individual ID. The salary is based on no of working days, extra working hrs, employee grade, paid leaves available and leaves consumed by the employee. For additional leaves consumed by the employee the salary will get deducted based on daily wages calculated. Salary for grade-1 employee is Rs.10, 000 and grade-2 is Rs.15, 000 monthly. Each employee can apply for the leave in advance.
2. The EmployeeAccounts class should support the employee salary calculations as per the above norms.
3. Define the test cases to match the above requirements to test with different employee criteria and accordingly update/modify the EmployeeAccounts and Employee classes.
4. Assume suitable additional data.
5. Refactor the code and make sure to make all the tests success.
6. Test the Employee salary in absence of EmployeeAccounts by using mock object.

Development setup: Windows 7/10 with JDK1.8, Eclipse-Jee-Photon

Steps for implementation

1. Define the test classes in test package with test methods to exercise the tests for EmployeeAccounts and Employee functional behaviour.
2. Define domain classes in com.company package.
3. Run the tests and add required domain code for the tests to succeed
4. Refactor the code and Repeat the steps of tests.

Steps for the TDD cycle

1. Define the test
2. Run the test.
3. Let it fail first time. The Red bar will be shown
4. Apply the required minimum implementation code to make the test succeed i.e. Green bar
5. Fake the implementation
6. Write obvious code for implementation
7. Apply generalized approach when the code becomes duplicate and dependent(i.e. Triangulation)
8. Run the test to see the green bar.
9. Run all the tests to check the success of all the tests.
10. Add more test methods to test more behaviour of account
11. Refactor the code to avoid duplicate code and future updation easier.
12. Run all the tests again to verify the success of refactoring.

Apply Code Refactoring

1. Remove the common duplicate code across different types of account and abstract it into a top level abstraction of Account.

2. Run all the tests and make all of them succeed.

Separate the process of object creation and usage.

1. A factory should return the Employee and EmployeeAccounts type objects to the program instead of creating them in the programs.
2. Define test class to test the object creation from a factory class by specifying type of Employee.
3. Define EmployeeAccountsFactory class with required functions.
4. Test the factory with different types of employee for success.
5. Run all the test methods to verify the success.

******