Chapter 1

general understanding about writing Unit Tests

1.1 Why We Use the `.spec` Suffix and Component Name in Test Files

When writing tests in Jest, it's common practice to use the `.spec.ts` or `.spec.js` suffix in the filename. This convention helps in clearly identifying the files that contain test cases. The name of the component or class being tested is typically included in the filename (e.g., `ContactClass.spec.ts`). This ensures that test files are organized and easily identifiable, which is crucial in large codebases. Using this naming convention also allows Jest to automatically find and run the test files without any additional configuration.

1.2 Example: Testing the `ContactClass` Component

Below is a series of test cases for a class called `ContactClass`. These examples will help you understand the structure of Jest tests and how you can effectively write your own.

```
import ContactClass from './contact';

describe('Contact class tests', () => {
  let contact: ContactClass | null; // Declares the
      contact variable as a ContactClass type

  beforeEach(() => {
    // Executes beforeEach function before each test
        case
    contact = new ContactClass();
  });

  it('should have a valid constructor', () => {
    // Tests that the contact is not null
    expect(contact).not.toBeNull();
  });

  it('should set name correctly through constructor',
     () => {
    contact = new ContactClass('barthauer');
    expect(contact.name).toEqual('barthauer');
  });

  it('should get and set id correctly', () => {
    contact!.id = 1;
    expect(contact!.id).toEqual(1);
  });

  it('should get and set name correctly', () => {
    contact!.name = 'barthauer';
    expect(contact!.name).toEqual('barthauer');
  });

  it('should get and set email correctly', () => {
    (contact as any)!.email = 'barthauer@barthauer.de
        '; // Casting to any to test dynamic property
    expect((contact as any)!.email).toEqual('
        barthauer@barthauer.de');
  });

  afterEach(() => {
    // Executes afterEach function after each test
        case
    contact = null;
  });
});
```

Listing 1.1: Test cases for the `ContactClass`.

1.3 Understanding the Structure of Jest Tests

1.3.1 `describe` Block

The `describe` block is used to group related tests together. In the example above, all tests related to `ContactClass` are grouped under a single `describe` block named `'Contact class tests'`.

1.3.2 `beforeEach` Function

The `beforeEach` function runs before each test case (`it` block). It's typically used for setting up the environment needed for the tests. In the example, `beforeEach` is used to initialize the `contact` variable before every test.

If you have multiple `beforeEach` functions at different levels (nested `describe` blocks), they will be executed in order from the outermost to the innermost block, ensuring proper setup for each scope.

1.3.3 `it` Blocks

Each `it` block represents an individual test case. The string argument describes what the test is verifying. For instance, `'should have a valid constructor'` tests whether the `contact` object is successfully created.

Multiple `it` blocks can be used to test different aspects of the class, like checking the constructor, property setters and getters, and other methods.

1.3.4 `afterEach` Function

The `afterEach` function runs after each test case. It's used for cleanup tasks, such as resetting variables or states. In this case, the `contact` variable is set to `null` after each test, ensuring no test case affects others.

1.3.5 Dynamic Properties

In the test case for setting and getting the email, the property `email` is dynamically added and tested using type casting (`as any`). This demonstrates how you can test properties or methods that might not be explicitly defined in the class but are still part of the test scenarios.

1.3.6 Nested `describe` Blocks (Optional)

You can have nested `describe` blocks to further organize your tests, especially when dealing with complex components with multiple methods. Each nested block can have its own `beforeEach` and `afterEach` functions, allowing for more granular setup and teardown processes.

1.4 Summary

By organizing your tests using `describe`, `it`, `beforeEach`, and `afterEach`, you create a clear, maintainable, and effective testing structure. Each test is isolated, ensuring that one test's setup or outcome doesn't affect another. The use of `.spec` suffixes and component names in filenames further enhances the readability and manageability of your test suite.