TDD (Test Driven Development) feat.Jest

OOP

Object-oriented programming (OOP) is a popular programming paradigm in which code is organized around objects instead of functions. This approach has the added advantage of helping us follow DRY practices.

TDD

Test-driven development (TDD) is the practice of writing tests before writing code. Here's a snapshot of the TDD process:

- Write a failing test before writing any code.
- Write the minimal amount of code necessary to make the test pass.
- Refactor the code to make the test pass.

Jest

- We can use Jest to test our code.
- **Jest** is a testing framework that was created and maintained by Facebook—and it's growing in popularity. Its syntax is easy to understand, and it is extremely popular in Node.js apps.

jest directory structure

- **tests**/: contains the tests for the code.
- fileName.test.js: contains the tests for the fileName.js file. `test.js` is a convention for test files.
- Sample test to test fucntion sum of number

```
describe('sum', () => {
  it('should return the sum of two numbers', () => {
    expect(sum(1, 2)).toBe(3);
  });
});
```

expect outcome methods

- `any`: The `any` matcher is used to test if a value is anything other than `null` or `undefined`.
- `toBe`: check result to be equal to expected value.
- `isGreaterThan`: check result to be greater than expected value.
- `isLessThan`: check result to be less than expected value.
- `toEqual`: check result to be equal to expected value.
- `toBeTruthy`: check result to be truthy.
- `toBeLessThanOrEqual`: check result to be less than or equal to expected value.
- `toBeGreaterThanOrEqual`: check result to be greater than or equal to expected value.
- `toBeCloseTo`: check result to be close to expected value.
- `toBeInstanceOf`: check result to be instance of expected value.
- `toBeNull`: check result to be null.
- `toBeUndefined`: check result to be undefined.
- `toBeDefined`: check result to be defined.
- `toBeNaN`: check result to be NaN.
- `toBeFalsy`: check result to be falsy.
- `toContain`: check result to contain expected value.
- `toMatch`: check result to match expected value.
- `toHaveLength`: check result to have length of expected value.
- `toHaveProperty`: check result to have property of expected value.
- `arrayContaining`: check result to contain expected value.

Mock

- **Mocks** allow us to fake assumed data, which allows the test at hand to focus only on the logic it cares about.
- we store mock files in **mocks** directory.

running test is jest

- `npm run test` : run test.
- `npm run test ExampleFileName` : run test for ExampleFileName.
- `npm run test -- --watch` : run test and watch for changes.
- `npm run test -- --coverage` : run test and generate coverage, coverage is a tool that generates a report of the code coverage of your project.

Constructors

- Constructors are functions that are used to create new objects.
- we can use `new` keyword to create new object.
- the name of the constructor function is preferre to be capitalized. e.g. `Person`
- constructor contains all the properties of the object, e.g: A Car
- `this` is a reference to the object being created.
- `new` keyword is used to create a new object.

```
function Car(make='', model, year) {
  this.make = make; // will be set to default '' id no make is passed in
  this.model = model;
  this.year = year;
}
let car1 = new Car('Toyota', 'Corolla', 2000);
```

SOLID Principles

- Single Responsibility Principle: A class should have a single responsibility.
- the scope of the function should be limited to the class and contain only the code that is required to perform the task.

Prototype

- Prototype objects simply inherit the method from the constructor rather than having their own instances of that method.
- Prototype is a way to share methods and properties between objects.
- Why use it? We don't want to create a new object for every instance of the class. We want to share the same methods and properties.
- so we write the function once and then use it in multiple objects.