

Functional Diagram Practice & More on Testing

Have out Paper and Pencil *or an iPad/Tablet+Stylus* - Laptops placed away! Bags under seats!

Only use Zones ABC. No DEF!

Kris Jordan / September 6, 2024 / COMP423 / Class 07

Diagram the following code listing

```
1  interface VoidFn<T> {
2  |   (arg: T): void;
3  | }
4
5  export const forEach = <T>(items: T[], f: VoidFn<T>) => {
6  |   for (const item of items) {
7  |     f(item);
8  |   }
9  | };
10
11 const letters = ["a", "b"];
12 forEach(letters, (item) => {
13 |   console.log(item);
14 | });
```

```
1 interface VoidFn<T> {
2   | (arg: T): void;
3 }
4
5 export const forEach = <T>(items: T[], f: VoidFn<T>) => {
6   | for (const item of items) {
7     |   f(item);
8   | }
9 };
10
11 const letters = ["a", "b"];
12 forEach(letters, (item) => {
13   | console.log(item);
14 });
```


How would you unit test forEach?

It's a void function...

```
1 interface VoidFn<T> {
2   |   (arg: T): void;
3   | }
4
5   export const forEach = <T>(items: T[], f: VoidFn<T>) => {
6   |   for (const item of items) {
7   |     f(item);
8   |   }
9   | };
10
11  const letters = ["a", "b"];
12  forEach(letters, (item) => {
13  |   console.log(item);
14  | });
```

Spying and mocking is useful for a behavioral dependency like f. We need to know f gets called (*spying*) without worrying it has an effect (*mocking*).

```
1  interface VoidFn<T> {
2  |   (arg: T): void;
3  | }
4
5  export const forEach = <T>(items: T[], f: VoidFn<T>) => {
6  |   for (const item of items) {
7  |     f(item);
8  |   }
9  | };
10
11  const letters = ["a", "b"];
12  forEach(letters, (item) => {
13  |   console.log(item);
14  | });
```



Mocking Functions/Methods with `jest.fn()`

`jest.fn()` returns a spied mock to substitute for dependent functions/methods

```
describe("forEach", () => {
  it("should call the function for each item in the array", () => {
    const mockFn = jest.fn();
    const items = ["a", "b", "c"];

    forEach(items, mockFn);

    expect(mockFn).toHaveBeenCalledTimes(3);
    for (let i = 0; i < items.length; i++) {
      expect(mockFn).toHaveBeenNthCalledWith(i + 1, items[i]);
    }
  });
});
```

Mocking Functions/Methods with `jest.fn()`

`jest.fn()` returns a spied mock to substitute for dependent functions/methods

```
describe("forEach", () => {  
  it("should call the function for each item in the array", () => {  
    const mockFn = jest.fn();  
    const items = ["a", "b", "c"];  
  
    forEach(items, mockFn);  
  
    expect(mockFn).toHaveBeenCalledTimes(3);  
    for (let i = 0; i < items.length; i++) {  
      expect(mockFn).toHaveBeenNthCalledWith(i + 1, items[i]);  
    }  
  });  
});
```

Establishing the spied mock.

Unit under test is given the mock as a dependency.

Asserting expected, spied behavior

TypeScript and JavaScript's Built-in Array Methods

forEach, map, filter

- **forEach** is a void method that calls a given function for each item in an array.

```
["a", "b"].forEach((letter) => { console.log(letter); });
```

- **map** is a method that calls a function for each item in an array and returns a new array with the results of each of the function calls.

```
[1, 2].map((x) => { return x * 2; }); // Returns a new array [2, 4]
```

- **filter** is a method that calls a function for each item in an array, if the return value is true (or truthy), the item is added to a returned array

```
[1, 2, 3].filter((x) => { return x % 2 == 1; }); // Returns [1, 3]
```


How many "units" of testable code are there?

For each unit, are there any *dependencies* you need to isolate to unit test?

```
export interface Powered {
  on(): void;
  // off(): void; -- This is obviously missing...
}

export class PowerStrip {
  constructor(private devices: Powered[]) {}

  on() {
    this.devices.forEach((device) => device.on());
  }
}

export class Lamp implements Powered {
  on() {
    console.log("Lamp on");
  }
}
```

Unit under test:

```
export class PowerStrip {
  constructor(private devices: Powered[]) {}

  on() {
    this.devices.forEach((device) => device.on());
  }
}
```

Mocking dependencies:

```
describe("PowerStrip", () => {
  let mocks: Powered[];

  // Setup mock Powered objects whose on() uses jest.fn()
  beforeEach(() => (mocks = [{ on: jest.fn() }, { on: jest.fn() }]]));

  afterEach(() => jest.restoreAllMocks());

  it("on() should call all devices' on()", () => {
    const strip = new PowerStrip(mocks);

    // Unit under integration test
    strip.on();

    mocks.forEach((mock) => expect(mock.on).toHaveBeenCalled());
  });
});
```

Diagram the following code listing

```
1   export const f = (x: number) => {
2     return (y: number) => {
3       return (z: number) => {
4         return x + y + z;
5       };
6     };
7   };
8
9   const a = f(1);
10  const b = a(2);
11  const c = b(3);
12  console.log(c);
```

```
1  export const f = (x: number) => {
2      return (y: number) => {
3          return (z: number) => {
4              return x + y + z;
5          };
6      };
7  };
8
9  const a = f(1);
10 const b = a(2);
11 const c = b(3);
12 console.log(c);
```