TDD IS NOT A STUPID IDEA

IT'S BRILLIANT!

BY @DRPICOX

TDD

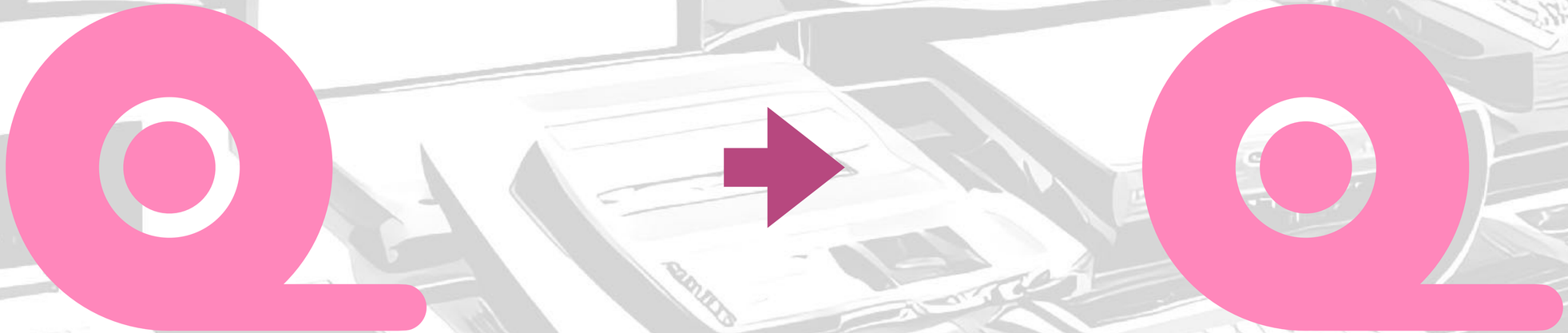
1. Write the test
2. Write the code
3. Cleanup
4. GOTO 1.

https://youtu.be/cGuTmOUdFbo

xUNIT

1. Input

2. Output



3. Code

«If an idea is good,
and turn out to be true,
somebody else will have done it;

…

...

but,

...

...

if an idea is stupid,
you have a chance that nobody else
is dumb enough to try it,
then,
if it happens to work,
you really have something.»

KENT IS CRAZY!

* Actually we do not know if Werd Cunningham said that

STACK*

- push
- pop
- isEmpty

* **Reenactment**
** he had no JS array, and we won't use it

```
// Stack.js
```

```
// Stack.spec.js
```

```
// Stack.js
```

```
// Stack.spec.js
import { Stack } from "./stack";

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});
```

```
// Stack.js
```

```
// Stack.spec.js
import { Stack } from "./stack";

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});
```

```
// Stack.js
```

TypeError: Stack is not a constructor

```
// Stack.spec.js
import { Stack } from "./stack";

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});
```

```
// Stack.js
export class Stack {}
```

TypeError: stack.isEmpty is not a function

```
// Stack.spec.js
import { Stack } from "./stack";

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});
```

```
// Stack.js
export class Stack {
  isEmpty() {}
}
```

Expected: true, Received: undefined

```javascript
// Stack.spec.js
import { Stack } from "./stack";

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});
```

```javascript
// Stack.js
export class Stack {
  isEmpty() {
    return true;
  }
}
```

```javascript
// Stack.spec.js
import { Stack } from "./stack";

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  const stack = new Stack();
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});
```

```javascript
// Stack.js
export class Stack {
  isEmpty() {
    return true;
  }
}
```

```
TypeError: stack.push is not a function
```

```javascript
// Stack.spec.js
import { Stack } from "./stack";

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  const stack = new Stack();
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});
```

```javascript
// Stack.js
export class Stack {
  isEmpty() {
    return true;
  }

  push() {}
}
```

Expected: false, Received: true

```javascript
// Stack.spec.js
import { Stack } from "./stack";

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test.skip("Adds one element", () => {
  const stack = new Stack();
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});
```

```javascript
// Stack.js
export class Stack {
  isEmpty() {
    return true;
  }

  push() {}
}
```

```javascript
// Stack.spec.js
import { Stack } from "./stack";

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test.skip("Adds one element", () => {
  const stack = new Stack();
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;

  isEmpty() {
    return this.#empty;
  }

  push() {}
}
```

```javascript
// Stack.spec.js
import { Stack } from "./stack";

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  const stack = new Stack();
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;

  isEmpty() {
    return this.#empty;
  }

  push() {}
}
```

Expected: false, Received: true

```javascript
// Stack.spec.js
import { Stack } from "./stack";

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  const stack = new Stack();
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;

  isEmpty() {
    return this.#empty;
  }

  push() {
    this.#empty = false;
  }
}
```

```javascript
// Stack.spec.js
import { Stack } from "./stack";

let stack;
beforeEach(() => {
  stack = new Stack();
});

test("Emtpy stack", () => {
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;

  isEmpty() {
    return this.#empty;
  }

  push() {
    this.#empty = false;
  }
}
```

```javascript
// Stack.spec.js
import { Stack } from "./stack";

let stack;
beforeEach(() => {
  stack = new Stack();
});

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;

  isEmpty() {
    return this.#empty;
  }

  push() {
    this.#empty = false;
  }
}
```

```
TypeError: stack.pop is not a function
```

```javascript
// Stack.spec.js
import { Stack } from "./stack";

let stack;
beforeEach(() => {
  stack = new Stack();
});

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;

  isEmpty() {
    return this.#empty;
  }

  push() {
    this.#empty = false;
  }

  pop() {}
}
```

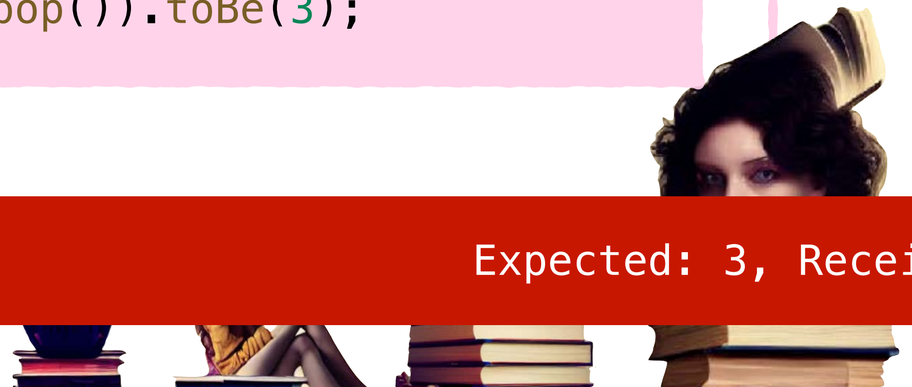
Expected: 1, Received: undefined

```javascript
// Stack.spec.js
import { Stack } from "./stack";

let stack;
beforeEach(() => {
  stack = new Stack();
});

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;

  isEmpty() {
    return this.#empty;
  }

  push() {
    this.#empty = false;
  }

  pop() {
    return 1;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});

test("Pops other pushed value", () => {
  stack.push(3);
  expect(stack.pop()).toBe(3);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;

  isEmpty() {
    return this.#empty;
  }

  push() {
    this.#empty = false;
  }

  pop() {
    return 1;
  }
}
```



Expected: 3, Received: 1

```javascript
// ...Stack.spec.js...

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});

test.skip("Pops other pushed value", () => {
  stack.push(3);
  expect(stack.pop()).toBe(3);
});
```

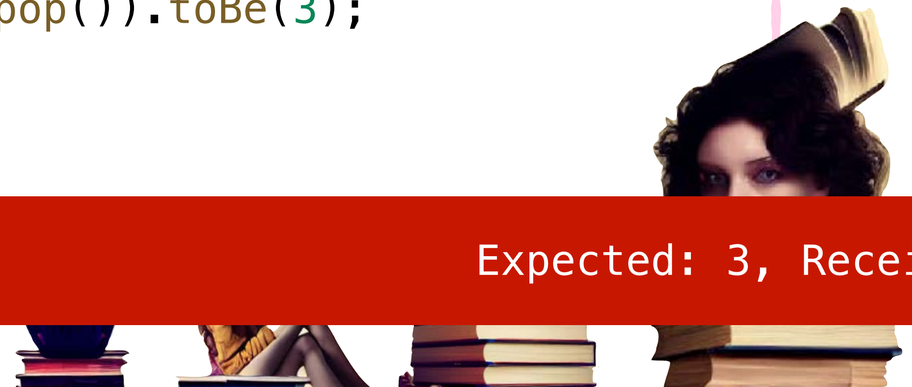
```javascript
// Stack.js
export class Stack {
  #empty = true;

  isEmpty() {
    return this.#empty;
  }

  push() {
    this.#empty = false;
  }

  pop() {
    return 1;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});

test.skip("Pops other pushed value", () => {
  stack.push(3);
  expect(stack.pop()).toBe(3);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;
  #value = 1;

  isEmpty() {
    return this.#empty;
  }

  push() {
    this.#empty = false;
  }

  pop() {
    return this.#value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});

test("Pops other pushed value", () => {
  stack.push(3);
  expect(stack.pop()).toBe(3);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;
  #value = 1;

  isEmpty() {
    return this.#empty;
  }

  push() {
    this.#empty = false;
  }

  pop() {
    return this.#value;
  }
}
```



Expected: 3, Received: 1

```javascript
// ...Stack.spec.js...

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});

test("Pops other pushed value", () => {
  stack.push(3);
  expect(stack.pop()).toBe(3);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;
  #value = 1;

  isEmpty() {
    return this.#empty;
  }

  push(value) {
    this.#value = value;
    this.#empty = false;
  }

  pop() {
    return this.#value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});

test("Pops other pushed value", () => {
  stack.push(3);
  expect(stack.pop()).toBe(3);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;
  #value;

  isEmpty() {
    return this.#empty;
  }

  push(value) {
    this.#value = value;
    this.#empty = false;
  }

  pop() {
    return this.#value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});

test("Pops other pushed value", () => {
  stack.push(3);
  expect(stack.pop()).toBe(3);
});
```

```javascript
// Stack.js
export class Stack {
  #empty = true;
  #value;

  isEmpty() {
    return this.#value == null;
  }

  push(value) {
    this.#value = value;
    this.#empty = false;
  }

  pop() {
    return this.#value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Emtpy stack", () => {
  const stack = new Stack();
  expect(stack.isEmpty()).toBe(true);
});

test("Adds one element", () => {
  stack.push(1);
  expect(stack.isEmpty()).toBe(false);
});

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});

test("Pops other pushed value", () => {
  stack.push(3);
  expect(stack.pop()).toBe(3);
});
```

```javascript
// Stack.js
export class Stack {
  #value;

  isEmpty() {
    return this.#value == null;
  }

  push(value) {
    this.#value = value;
  }

  pop() {
    return this.#value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});

test("Pops other pushed value", () => {
  stack.push(3);
  expect(stack.pop()).toBe(3);
});

test("Pop after push isEmpty", () => {
  stack.push(1);
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});
```

```javascript
// Stack.js
export class Stack {
  #value;

  isEmpty() {
    return this.#value == null;
  }

  push(value) {
    this.#value = value;
  }

  pop() {
    return this.#value;
  }
}
```

Expected: true, Received: false

```javascript
// ...Stack.spec.js...

test("Pops the pushed value", () => {
  stack.push(1);
  expect(stack.pop()).toBe(1);
});

test("Pops other pushed value", () => {
  stack.push(3);
  expect(stack.pop()).toBe(3);
});

test("Pop after push isEmpty", () => {
  stack.push(1);
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});
```

```javascript
// Stack.js
export class Stack {
  #value;

  isEmpty() {
    return this.#value == null;
  }

  push(value) {
    this.#value = value;
  }

  pop() {
    const value = this.#value;
    this.#value = null;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Pops other pushed value", () => {
  stack.push(3);
  expect(stack.pop()).toBe(3);
});

test("Pop after push isEmpty", () => {
  stack.push(1);
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});

test("Double push, single pop", () => {
  stack.push(1);
  stack.push(2);
  expect(stack.pop()).toBe(2);
});
```

```javascript
// Stack.js
export class Stack {
  #value;

  isEmpty() {
    return this.#value == null;
  }

  push(value) {
    this.#value = value;
  }

  pop() {
    const value = this.#value;
    this.#value = null;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Pop after push isEmpty", () => {
  stack.push(1);
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});

test("Double push, single pop", () => {
  stack.push(1);
  stack.push(2);
  expect(stack.pop()).toBe(2);
});

test("Double push, double pop", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  expect(stack.pop()).toBe(1);
});
```

```javascript
// Stack.js
export class Stack {
  #value;

  isEmpty() {
    return this.#value == null;
  }

  push(value) {
    this.#value = value;
  }

  pop() {
    const value = this.#value;
    this.#value = null;
    return value;
  }
}
```

Expected: 1, Received: null

```javascript
// ...Stack.spec.js...

test("Pop after push isEmpty", () => {
  stack.push(1);
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});

test("Double push, single pop", () => {
  stack.push(1);
  stack.push(2);
  expect(stack.pop()).toBe(2);
});

test.skip("Double push, double pop", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  expect(stack.pop()).toBe(1);
});
```

```javascript
// Stack.js
export class Stack {
  #value;

  isEmpty() {
    return this.#value == null;
  }

  push(value) {
    this.#value = value;
  }

  pop() {
    const value = this.#value;
    this.#value = null;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Pop after push isEmpty", () => {
  stack.push(1);
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});

test("Double push, single pop", () => {
  stack.push(1);
  stack.push(2);
  expect(stack.pop()).toBe(2);
});

test.skip("Double push, double pop", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  expect(stack.pop()).toBe(1);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = value;
  }

  pop() {
    const value = this.#top;
    this.#top = null;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Pop after push isEmpty", () => {
  stack.push(1);
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});

test("Double push, single pop", () => {
  stack.push(1);
  stack.push(2);
  expect(stack.pop()).toBe(2);
});

test.skip("Double push, double pop", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  expect(stack.pop()).toBe(1);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value };
  }

  pop() {
    const { value } = this.#top;
    this.#top = null;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Pop after push isEmpty", () => {
  stack.push(1);
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});

test("Double push, single pop", () => {
  stack.push(1);
  stack.push(2);
  expect(stack.pop()).toBe(2);
});

test("Double push, double pop", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  expect(stack.pop()).toBe(1);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value };
  }

  pop() {
    const { value } = this.#top;
    this.#top = null;
    return value;
  }
}
```

```
TypeError: Cannot destructure property 'value' of 'this[#top]'
```

```javascript
// ...Stack.spec.js...

test("Pop after push isEmpty", () => {
  stack.push(1);
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});

test("Double push, single pop", () => {
  stack.push(1);
  stack.push(2);
  expect(stack.pop()).toBe(2);
});

test("Double push, double pop", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  expect(stack.pop()).toBe(1);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Double push, single pop", () => {
  stack.push(1);
  stack.push(2);
  expect(stack.pop()).toBe(2);
});

test("Double push, double pop", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  expect(stack.pop()).toBe(1);
});

test('Pop an empty stack is null', () => {
  expect(stack.pop()).toBe(null);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

```
TypeError: Cannot destructure property 'value' of 'this[#top]'
```

```javascript
// ...Stack.spec.js...

test("Double push, single pop", () => {
  stack.push(1);
  stack.push(2);
  expect(stack.pop()).toBe(2);
});

test("Double push, double pop", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  expect(stack.pop()).toBe(1);
});

test('Pop an empty stack is null', () => {
  expect(stack.pop()).toBe(null);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    if (this.#top == null) return null;
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Double push, single pop", () => {
  stack.push(1);
  stack.push(2);
  expect(stack.pop()).toBe(2);
});

test("Double push, double pop", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  expect(stack.pop()).toBe(1);
});

test('Pop an empty stack is null', () => {
  expect(stack.pop()).toBe(null);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    if (this.isEmpty()) return null;
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Double push, double pop", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  expect(stack.pop()).toBe(1);
});

test('Pop an empty stack is null', () => {
  expect(stack.pop()).toBe(null);
});

test("Pop empty remains empty", () => {
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    if (this.isEmpty()) return null;
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test('Pop an empty stack is null', () => {
  expect(stack.pop()).toBe(null);
});

test("Pop empty remains empty", () => {
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});

test("Push-Pop x2 remains empty", () => {
  stack.push(1);
  stack.pop();
  stack.push(1);
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    if (this.isEmpty()) return null;
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Push-Pop x2 remains empty", () => {
  stack.push(1);
  stack.pop();
  stack.push(1);
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});

test("Push null is not empty", () => {
  stack.push(null);
  expect(stack.isEmpty()).toBe(false);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    if (this.isEmpty()) return null;
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Push twice pop twice empty", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    if (this.isEmpty()) return null;
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Pop twice isEmpty", () => {
  stack.pop();
  stack.pop();
  expect(stack.isEmpty()).toBe(true);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    if (this.isEmpty()) return null;
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Push after 2xPop", () => {
  stack.push(1);
  stack.pop();
  stack.pop();
  stack.push(2);
  expect(stack.pop()).toBe(2);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    if (this.isEmpty()) return null;
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Many push and pop", () => {
  stack.push(1);
  stack.push(2);
  stack.pop();
  stack.push(3);
  stack.push(4);
  stack.pop();
  stack.pop();
  expect(stack.pop()).toBe(1);
});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    if (this.isEmpty()) return null;
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

```javascript
// ...Stack.spec.js...

test("Make fail the Stack!!!", () => {

  // ???

});
```

```javascript
// Stack.js
export class Stack {
  #top;

  isEmpty() {
    return this.#top == null;
  }

  push(value) {
    this.#top = { value, next: this.#top };
  }

  pop() {
    if (this.isEmpty()) return null;
    const { value, next } = this.#top;
    this.#top = next;
    return value;
  }
}
```

«It was like cheating.

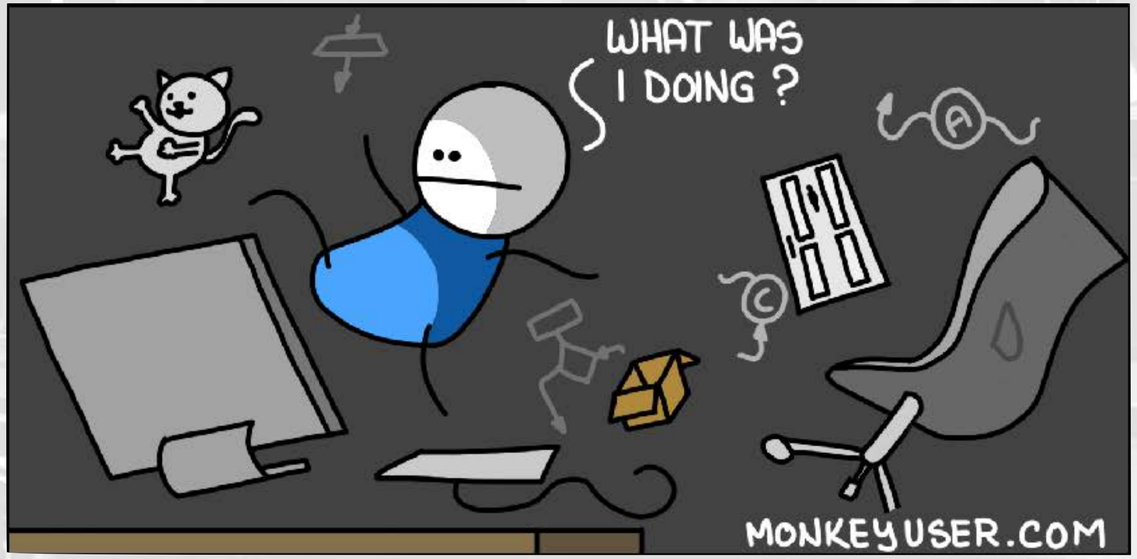
That heroic moment as a programmer,
when things are about to get out of control,
and then through the sheer force
of your intellect and your will,
you pull the order out of chaos.

That moment was gone.»

https://youtu.be/cGuTmOUdFbo


IT
DOESN'T SEEM LIKE
A STUPID IDEA

Simplifies Development

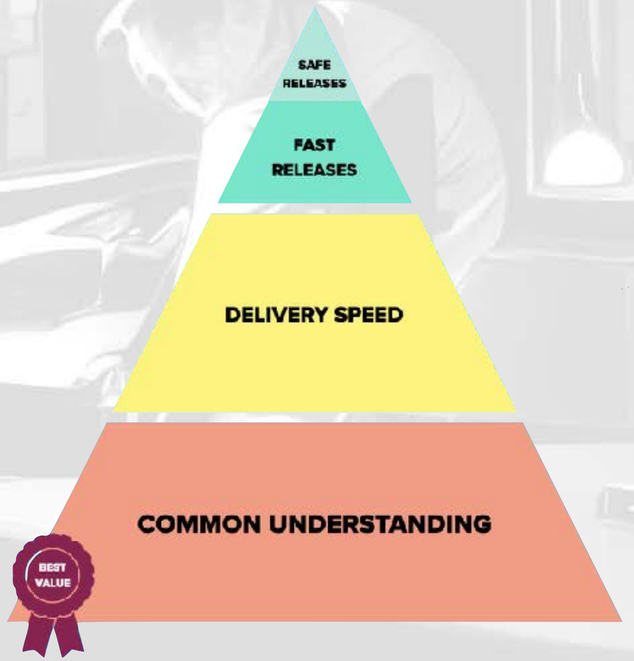


https://www.monkeyuser.com/2018/focus/

Smarter Architecture


Goal Oriented

Improves Communication



- Improved Quality
- Improved Design
- Reduces Cognitive Load
- Decouples Solution and Design
- Reduces Technical Debt
- Create Trustworthy tests
- Faster integration of new members
- Everything worked 30s ago
- Fast Feedback
- Bolder coders
- Less Debug
- Increased delivery speed
- Stops QA Cobra Effect
- Enables Continuous Delivery
- Tests are documentation
- Better understanding
- Better communication
- TDD is like cheating
- ...


IT'S BRILLIANT!


IT'S COUNTERINTUITIVE. AND BECAUSE OF THAT, IT'S HARD TO LEARN


BUT IT MAKES ME POWERFUL


THANKS!