Refactor Example: Replace Function With Command

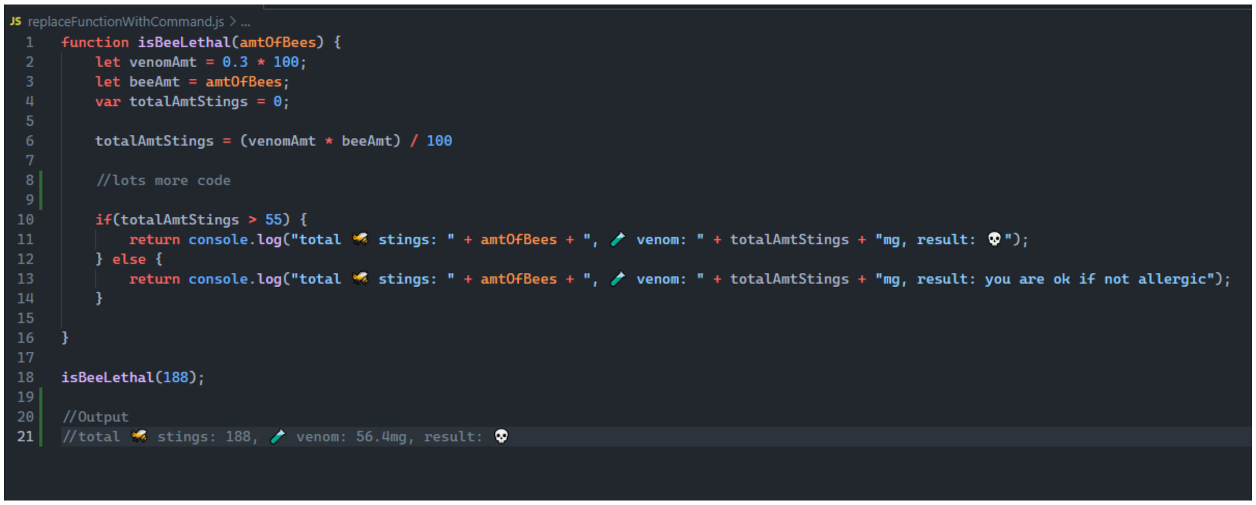
Starter code: https://github.com/carlosgonzalez96/replace-function-with-command

Code context: The code is simple; it just takes in a value that represents the number of bees and spits back some information. The info given back is the total amount of venom in milligrams and if that is a potent amount to kill someone.

What is Replace Function with command?

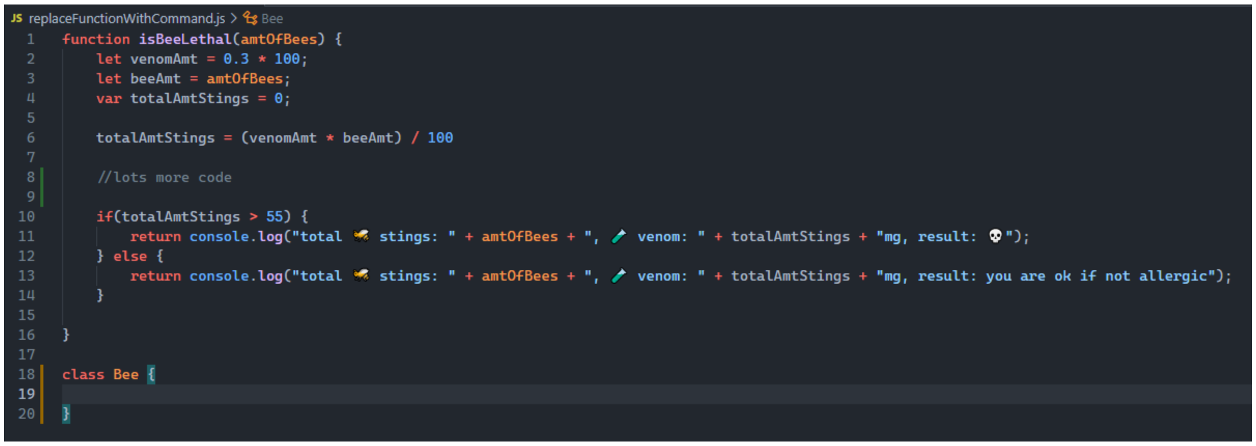
If you ever encountered a long function that returns some output, then you have encountered a code smell. Functions should be kept short and simple. There are multiple ways to refactor a long function but there are times when it is useful to encapsulate a function to its own object. That object is referred to a command object which is just a simply single method that only gets executed. The benefit of doing this is it gives better flexibility and control for the function expression.

Example: Here is a function that creates a few variables, calculates then, and returns a result based on the calculation. This refactoring technique works best if it reeks with a bunch of code that does a bunch of different calculations in a function.



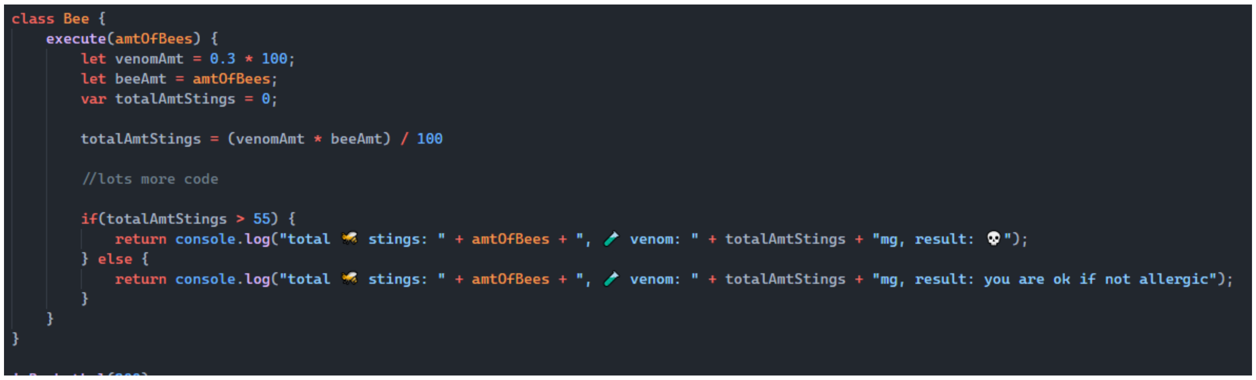
```
JS replaceFunctionWithCommand.js > ...
function isBeeLethal(amtOfBees) {
    let venomAmt = 0.3 * 100;
    let beeAmt = amtOfBees;
    var totalAmtStings = 0;

    totalAmtStings = (venomAmt * beeAmt) / 100

    //lots more code

    if(totalAmtStings > 55) {
        return console.log("total 🐝 stings: " + amtOfBees + ", 💉 venom: " + totalAmtStings + "mg, result: 💀");
    } else {
        return console.log("total 🐝 stings: " + amtOfBees + ", 💉 venom: " + totalAmtStings + "mg, result: you are ok if not allergic");
    }

}

isBeeLethal(188);

//Output
//total 🐝 stings: 188, 💉 venom: 56.4mg, result: 💀
```

We can start by creating an empty class for our function body code will live in.



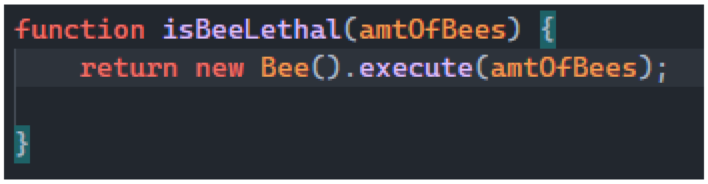
```
JS replaceFunctionWithCommand.js > Bee
function isBeeLethal(amtOfBees) {
    let venomAmt = 0.3 * 100;
    let beeAmt = amtOfBees;
    var totalAmtStings = 0;

    totalAmtStings = (venomAmt * beeAmt) / 100

    //lots more code

    if(totalAmtStings > 55) {
        return console.log("total 🐝 stings: " + amtOfBees + ", 💉 venom: " + totalAmtStings + "mg, result: 💀");
    } else {
        return console.log("total 🐝 stings: " + amtOfBees + ", 💉 venom: " + totalAmtStings + "mg, result: you are ok if not allergic");
    }

}

class Bee {

}
```

We then can create a new method called execute in the `Bee` class. In the `execute` method we can move all the content in the `isBeeLethal` function into the method.



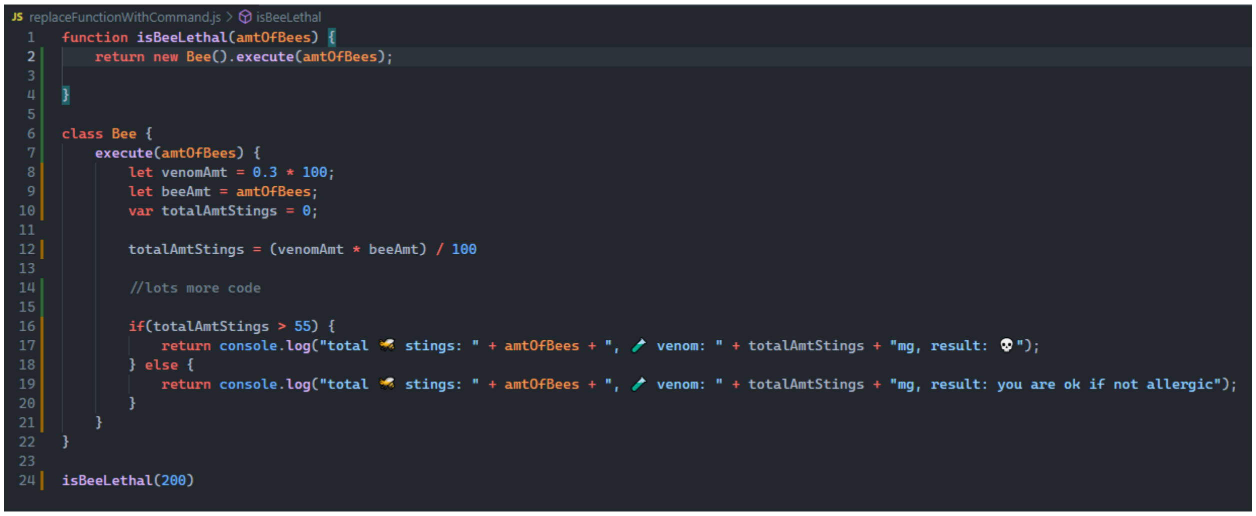
```
class Bee {
    execute(amtOfBees) {
        let venomAmt = 0.3 * 100;
        let beeAmt = amtOfBees;
        var totalAmtStings = 0;

        totalAmtStings = (venomAmt * beeAmt) / 100

        //lots more code

        if(totalAmtStings > 55) {
            return console.log("total 🐝 stings: " + amtOfBees + ", 💉 venom: " + totalAmtStings + "mg, result: 💀");
        } else {
            return console.log("total 🐝 stings: " + amtOfBees + ", 💉 venom: " + totalAmtStings + "mg, result: you are ok if not allergic");
        }
    }
}
```

We then can return an instance of Bee with the execute method in the original `isBeeLethal` function.



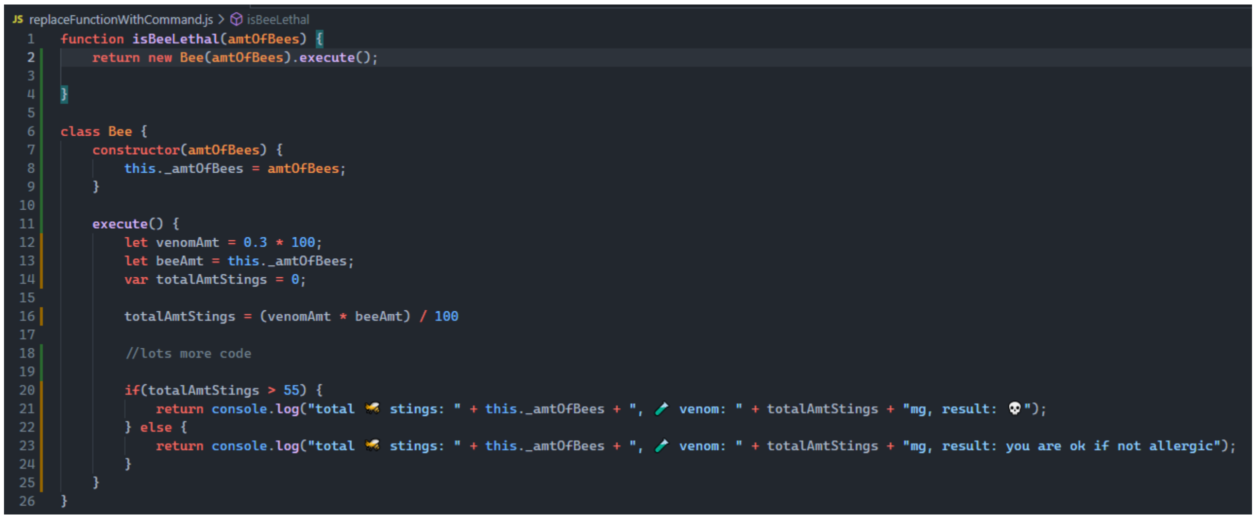
```
function isBeeLethal(amtOfBees) {
    return new Bee().execute(amtOfBees);

}
```

The code should now look like this:



```
JS replaceFunctionWithCommand.js > isBeeLethal
function isBeeLethal(amtOfBees) {
    return new Bee().execute(amtOfBees);

}

class Bee {
    execute(amtOfBees) {
        let venomAmt = 0.3 * 100;
        let beeAmt = amtOfBees;
        var totalAmtStings = 0;

        totalAmtStings = (venomAmt * beeAmt) / 100

        //lots more code

        if(totalAmtStings > 55) {
            return console.log("total 🐝 stings: " + amtOfBees + ", 💉 venom: " + totalAmtStings + "mg, result: 💀");
        } else {
            return console.log("total 🐝 stings: " + amtOfBees + ", 💉 venom: " + totalAmtStings + "mg, result: you are ok if not allergic");
        }
    }
}

isBeeLethal(200)
```

In our `Bee` class we see that the `execute` method is taking in a param. For our example we want to clear of any params in the `execute` method because it is better to pass it in with a constructor. Having the param go in the constructor will be much handy to manipulate if we get a more complicated parameter lifecycle. So, we will move the `amtOfBees` param into a constructor and set it equal with a new name that refers to the object. We will then rename any instances of `amtOfBees` in the execute method with the new name `this._amtOfBees`. The renaming exception will be in the isBeeLethal function. There we want to just move `amtOfBees` into the instance instead of the method call.



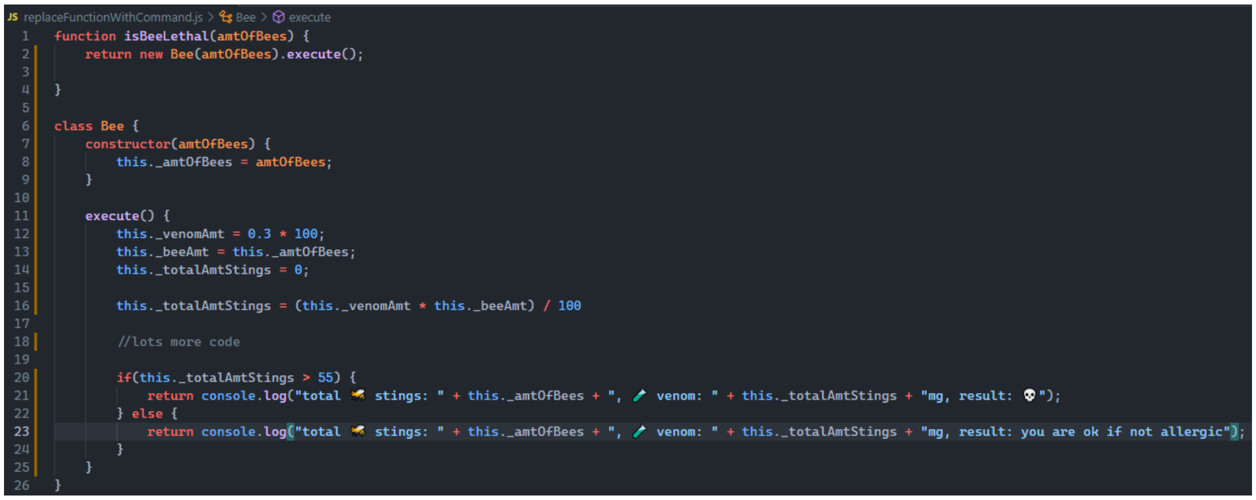
```javascript
function isBeeLethal(amtOfBees) {
    return new Bee(amtOfBees).execute();

}

class Bee {
    constructor(amtOfBees) {
        this._amtOfBees = amtOfBees;
    }

    execute() {
        let venomAmt = 0.3 * 100;
        let beeAmt = this._amtOfBees;
        var totalAmtStings = 0;

        totalAmtStings = (venomAmt * beeAmt) / 100

        //lots more code

        if(totalAmtStings > 55) {
            return console.log("total 🐝 stings: " + this._amtOfBees + ", 💉 venom: " + totalAmtStings + "mg, result: 💀");
        } else {
            return console.log("total 🐝 stings: " + this._amtOfBees + ", 💉 venom: " + totalAmtStings + "mg, result: you are ok if not allergic");
        }
    }
}
```

The refactoring is complete but like many refactoring's it makes the code available for more refactoring and modification.

Replace Function with Command allowed us to break a function that can be complicated or long. There is still more we could do to it!

After refactoring with replace function with command we start thinking about the variables. So, lets change the local variables in execute to field variables and its recommended to do this one variable at a time.

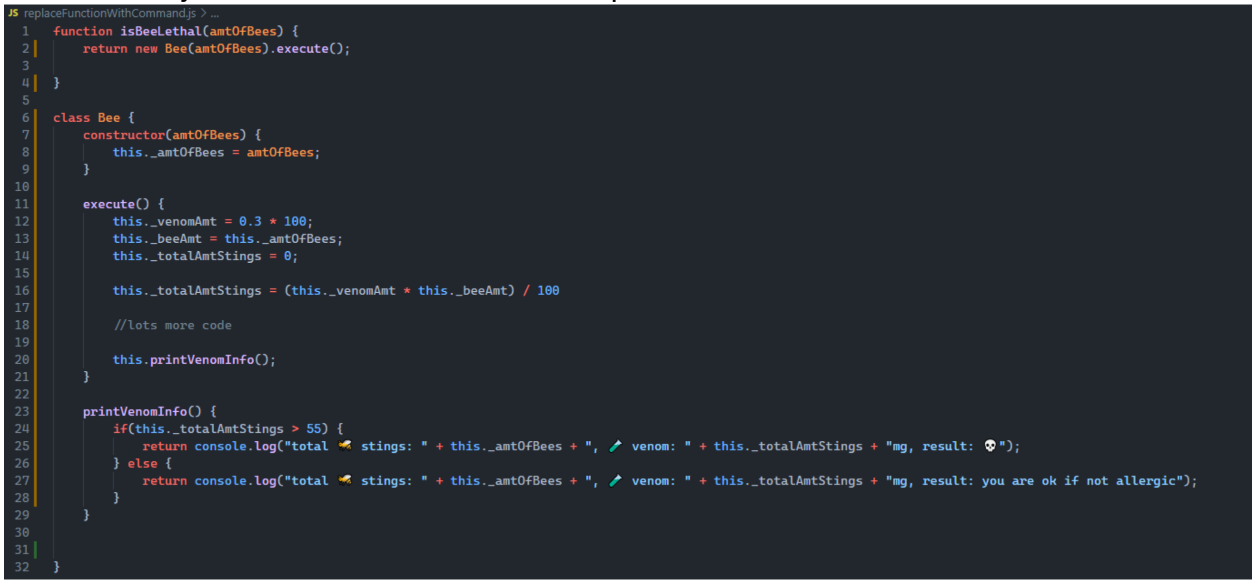
The result should like this:



```javascript
function isBeeLethal(amtOfBees) {
    return new Bee(amtOfBees).execute();

}

class Bee {
    constructor(amtOfBees) {
        this._amtOfBees = amtOfBees;
    }

    execute() {
        this._venomAmt = 0.3 * 100;
        this._beeAmt = this._amtOfBees;
        this._totalAmtStings = 0;

        this._totalAmtStings = (this._venomAmt * this._beeAmt) / 100

        //lots more code

        if(this._totalAmtStings > 55) {
            return console.log("total 🐝 stings: " + this._amtOfBees + ", 💉 venom: " + this._totalAmtStings + "mg, result: 💀");
        } else {
            return console.log("total 🐝 stings: " + this._amtOfBees + ", 💉 venom: " + this._totalAmtStings + "mg, result: you are ok if not allergic");
        }
    }
}
```

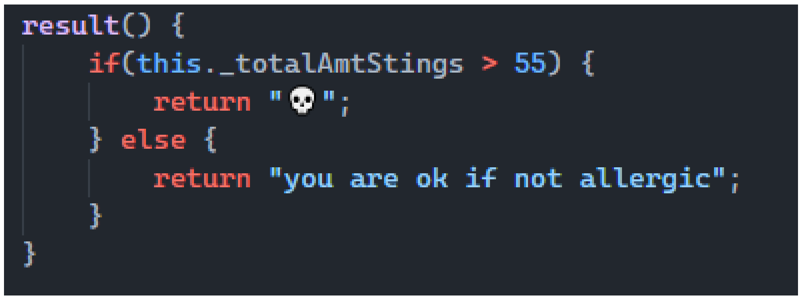
To clarify, field variables are only declared within the class so it can allow our variables to be seen by other methods inside the class. This can allow out code to be more flexible if we need add more things to our code.

We could additionally extract the if statement that returns our message into a new method and just call the function to clean up the structure a little better:



```
JS replaceFunctionWithCommand.js > ...
function isBeeLethal(amtOfBees) {
    return new Bee(amtOfBees).execute();

}

class Bee {
    constructor(amtOfBees) {
        this._amtOfBees = amtOfBees;
    }

    execute() {
        this._venomAmt = 0.3 * 100;
        this._beeAmt = this._amtOfBees;
        this._totalAmtStings = 0;

        this._totalAmtStings = (this._venomAmt * this._beeAmt) / 100

        //lots more code

        this.printVenomInfo();
    }

    printVenomInfo() {
        if(this._totalAmtStings > 55) {
            return console.log("total 🐝 stings: " + this._amtOfBees + ", 💉 venom: " + this._totalAmtStings + "mg, result: 💀");
        } else {
            return console.log("total 🐝 stings: " + this._amtOfBees + ", 💉 venom: " + this._totalAmtStings + "mg, result: you are ok if not allergic");
        }
    }


}
```

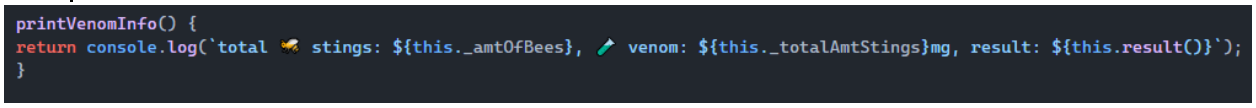
There is another smell, and that smell is duplication. In the conditional, notice that we must returns that print almost the same thing. The difference is at the end (result:). If the total amount of venom in a human body is more than 55mg then the result is at the end id *death*, otherwise a message that says *you are ok if not allergic*. We can return a one liner in the print function by applying some flocking rules and create a new function that determines and prints the ending result. To do so, create a new function called `result()` in the `Bee` class and have an if/else statement that reads:



```
result() {
    if(this._totalAmtStings > 55) {
        return "💀";
    } else {
        return "you are ok if not allergic";
    }
}
```

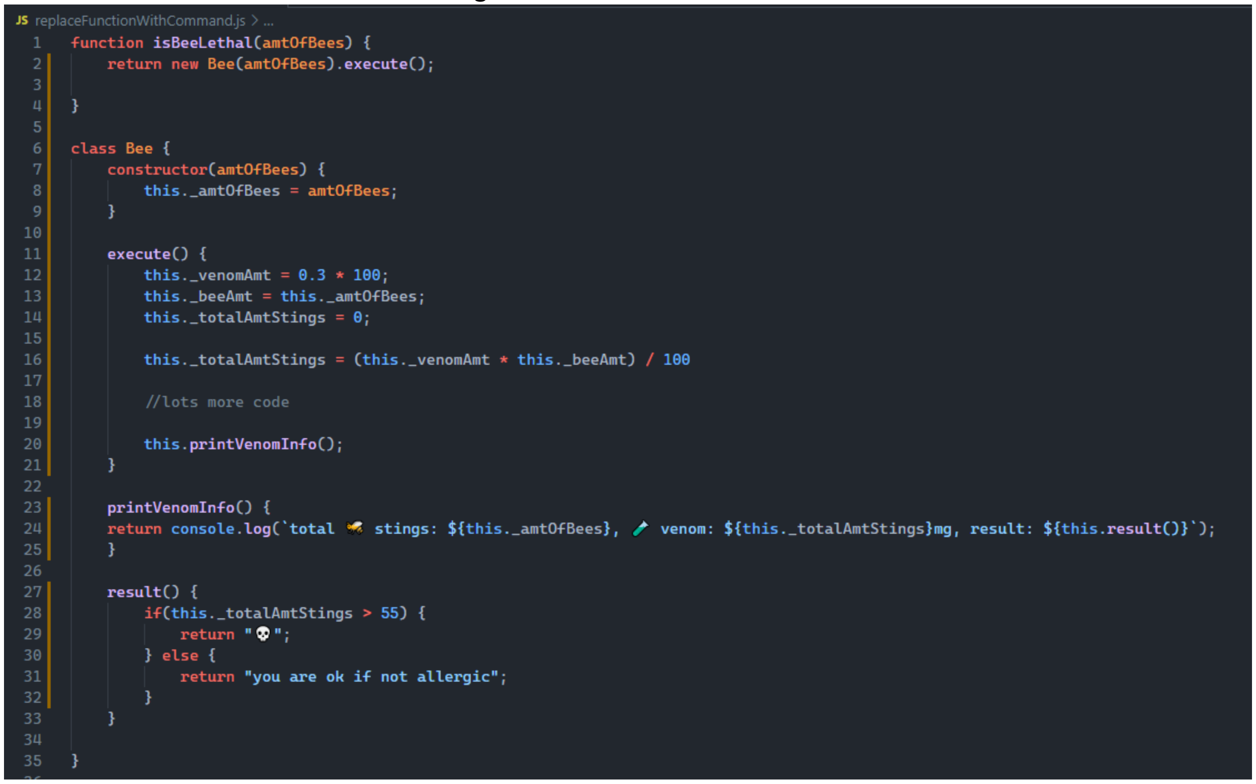
And in our `printVenomInfo`() function we can make a one liner return by changing the output to a template literal. This will allow us to embed our variables and the result() function to our string.

The print function should now look like this:



```
printVenomInfo() {
return console.log(`total 🐝 stings: ${this._amtOfBees}, 💉 venom: ${this._totalAmtStings}mg, result: ${this.result()}`);
}
```

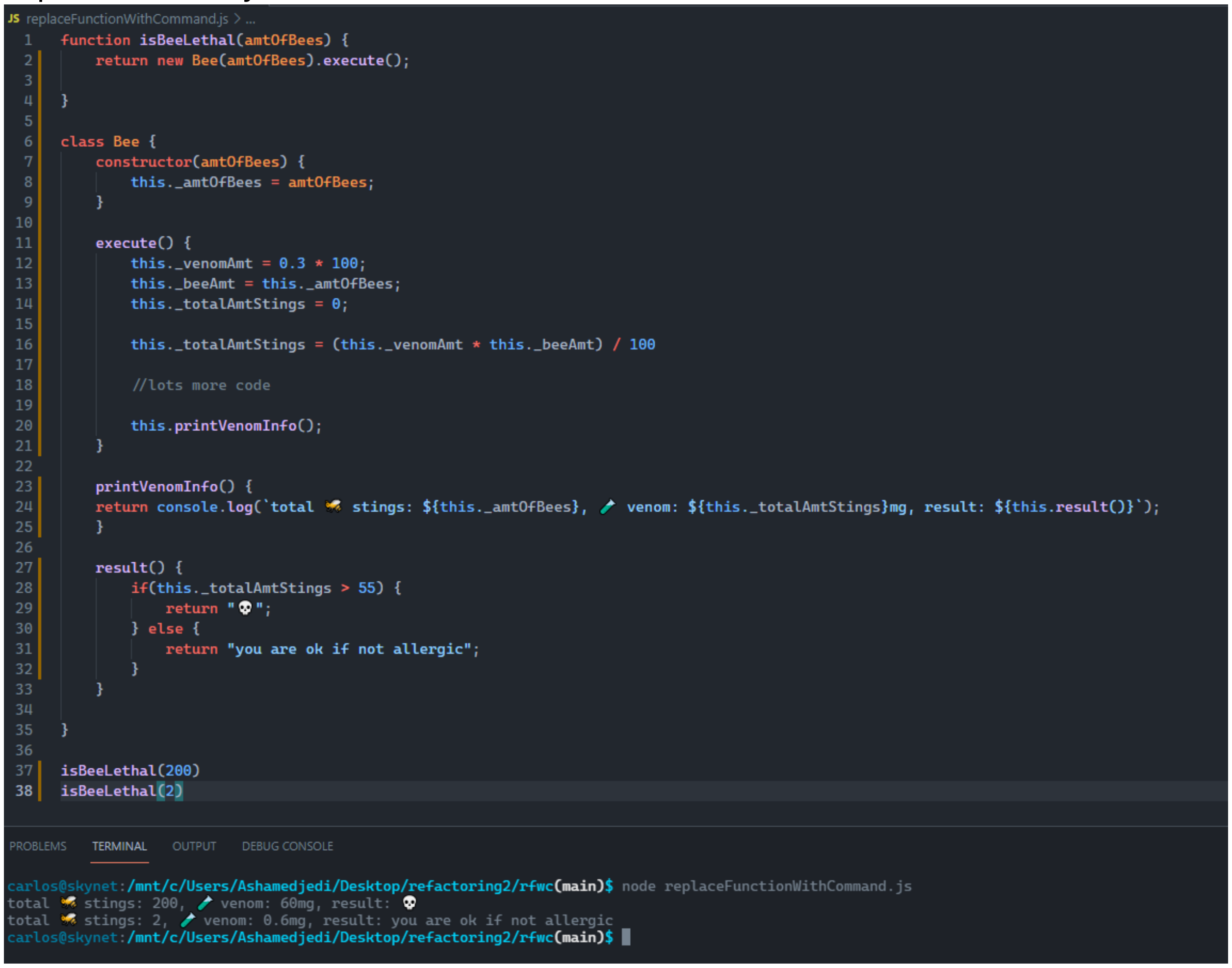
And that concludes the refactoring. The code base should now look this:



JS replaceFunctionWithCommand.js > ...

```javascript
function isBeeLethal(amtOfBees) {
    return new Bee(amtOfBees).execute();

}

class Bee {
    constructor(amtOfBees) {
        this._amtOfBees = amtOfBees;
    }

    execute() {
        this._venomAmt = 0.3 * 100;
        this._beeAmt = this._amtOfBees;
        this._totalAmtStings = 0;

        this._totalAmtStings = (this._venomAmt * this._beeAmt) / 100

        //lots more code

        this.printVenomInfo();
    }

    printVenomInfo() {
    return console.log(`total 🐝 stings: ${this._amtOfBees}, 💉 venom: ${this._totalAmtStings}mg, result: ${this.result()}`);
    }

    result() {
        if(this._totalAmtStings > 55) {
            return "💀";
        } else {
            return "you are ok if not allergic";
        }
    }

}
```

Do test the code out by calling the isBeeLethal() function and set a number of bees to explore how many bees can kill an adult human.



```javascript
function isBeeLethal(amtOfBees) {
    return new Bee(amtOfBees).execute();

}

class Bee {
    constructor(amtOfBees) {
        this._amtOfBees = amtOfBees;
    }

    execute() {
        this._venomAmt = 0.3 * 100;
        this._beeAmt = this._amtOfBees;
        this._totalAmtStings = 0;

        this._totalAmtStings = (this._venomAmt * this._beeAmt) / 100

        //lots more code

        this.printVenomInfo();
    }

    printVenomInfo() {
    return console.log(`total 🐝 stings: ${this._amtOfBees}, 🧪 venom: ${this._totalAmtStings}mg, result: ${this.result()}`);
    }

    result() {
        if(this._totalAmtStings > 55) {
            return "💀";
        } else {
            return "you are ok if not allergic";
        }
    }

}

isBeeLethal(200)
isBeeLethal(2)
```

```
carlos@skynet:/mnt/c/Users/Ashamedjedi/Desktop/refactoring2/rfwc(main)$ node replaceFunctionWithCommand.js
total 🐝 stings: 200, 🧪 venom: 60mg, result: 💀
total 🐝 stings: 2, 🧪 venom: 0.6mg, result: you are ok if not allergic
carlos@skynet:/mnt/c/Users/Ashamedjedi/Desktop/refactoring2/rfwc(main)$
```