Codigo Legacy

Indice

- ¿Que es el código legacy?
- Formas de trabajar con código legacy
  - Sprout
    - Sprout Method
    - Sprout Class
  - Wrap
    - Wrap Method
    - Wrap Class

¿Que es el código legacy?

Es código que heredamos de alguien mas o que es una versión de antigua del sistema o simplemente no entendemos el funcionamiento de este código

- No entendemos qué hace
- Codigo heredado
- Version antigua

```
function theUserIsHear() {
    const userList;
    const age;
    const userIsHere;

    if (userList != null  && userList != undefined) {
        userList.forEach(user => {
            if (user.age >= 18) {
                age = user.age;
            }

            if (user.address) {
                if (user.address.door) {
                    if (user.address.door.isOpen) {
                        userIsHere = true
                    }
                }
            }
        });
    }

    return userIsHere;
}
```

Imagen ilustrativa

DD

Doctrina del desarrollador

Sprout method

Sprout method

```
// Original method
getLastName() {
    const lastName = "Perez";
    return lastName;
}

// Same original method, but with new method to lower lastName
getLastName() {
    const lastName = "Perez";
    lastName = getLastNameToLowerCase(lastName);
    return lastName;
}

// New functionality
getLastNameToLowerCase(lastName) {
    return lastName.toLowerCase();
}
```

Este enfoque implica la identificación del punto de inserción de la nueva funcionalidad, seguido de la creación de un nuevo método en ese punto para encapsular dicha funcionalidad adicional

Además nos permite testear la nueva funcionalidad de una manera más cómoda y encapsulada.

*Otro beneficio es que nos permite mantener la retrocompatibilidad cuando sea necesario. Al trasladar la lógica a otro método, ganamos flexibilidad para gestionar múltiples comportamientos

Sprout method

Ventajas

- Se separa claramente el código nuevo con el existente
- Se puede testear de manera unitaria el nuevo código
- Permite retrocompatibilidad de ser necesaria

Desventajas

- No se modifica el código actual
- No se realizan mejoras al código actual por que se crea código nuevo encapsulado del existente

  * Si se puede realizar una mejora al código existente siempre es recomendable crear una nueva tarea/PR, para separar responsabilidades y minimizar la cantidad de cambios. Esto ayuda en la revisión del PR y posibles bugs

Sprout class

Sprout class

```
class User {

    // Original method
    getLastName() {
        const lastName = "Perez";
        return lastName;
    }

    // Same original method, but with new method to lower lastName
    getLastName() {
        const lastName = "Perez";
        lastName = new UserV2().getLastNameToLowerCase(lastName);
        return lastName;
    }
}

class UserV2 {

    // New functionality
    getLastNameToLowerCase(lastName) {
        return lastName.toLowerCase();
    }
}
```

Este enfoque implica identificar los cambios necesarios y, en caso de que la clase sea demasiado grande y complicada de probar debido a sus dependencias, creamos una nueva clase específica para realizar las modificaciones necesarias. Luego, llamamos a esta nueva clase desde el código que deseamos modificar.

Sprout class

Ventajas

- Al crear una nueva clase resulta más sencillo testear el nuevo código
- Está completamente desacoplado del código original, por que estamos creando una nueva clase independiente de la original

Desventajas

- Estamos generando una nueva entidad que no es la original, lo cual debemos tener cuidado con su uso y nombrado

Wrap method

Wrap method

```
class User {

    // Original method and rename
    getLastNameOlder() {
        const lastName = "Perez";
        return lastName;
    }

    // New functionality
    getLastNameToLowerCase(lastName) {
        return lastName.toLowerCase();
    }

    // New method with the same name to old method
    getLastName() {
        const lastName = this.getLastNameOlder();
        lastName = this.getLastNameToLowerCase(lastName);
        return lastName;
    }
}
```

En este enfoque, creamos un nuevo método en la clase con el mismo nombre que el método heredado y renombramos el método heredado. En el nuevo método, agregamos llamadas tanto al método heredado como al nuevo método que contendrá la funcionalidad adicional. De esta manera, encapsulamos tanto la lógica existente como la nueva dentro de un método que conserva el mismo nombre

Wrap method

Ventajas

- Es una buena manera de implementar una nueva funcionalidad de forma testeada

Desventajas

- Podemos incurrir en un mal nombramiento del método renombrado, lo cual genera confusión a los demás
- Explícitamente hace que la nueva funcionalidad sea independiente de la funcionalidad existente

Wrap class

Wrap method

```
class User {

    // Original method
    getLastName() {
        const lastName = "Perez";
        return lastName;
    }
}


class UserV2 extends User {

    user;

    constructor(user) {
        this.user = user;
    }

    // Original method
    getLastName() {
        const lastName = this.user.getLastName();
        lastName = this.getLastNameToLowerCase(lastName);
        return lastName;
    }

    // New functionality
    getLastNameToLowerCase(lastName) {
        return lastName.toLowerCase();
    }
}
```

Usa el mismo enfoque que Wrap Method nada más que envuelve la lógica nueva y la vieja un una clase

*Comparte el concepto del patrón decorador

DD

Doctrina del desarrollador

Wrap method

Ventajas

- Estamos creando una nueva clase la cual está desacoplada del resto, esto nos permite testearlo de una manera más sencilla

Desventajas

- Se aumenta la complejidad por que estamos agregando una nueva clase
- Se debe mantener la firma de la clase que estamos envolviendo

Referencias

- **Working Effectively With Legacy Code:** https://biratkirat.medium.com/working-effectively-with-legacy-code-mechanics-of-change-part-ii-chapter-1-91f2129c3b72