NO HACER UN GETTER DE UNA LISTA COMO ATRIBUTO

Causas:

1. Violación del principio de encapsulación: Los getters suelen utilizarse para acceder a los valores de un atributo de un objeto. Sin embargo, cuando se trata de una lista, proporcionar un getter permite que el código externo modifique directamente la lista, lo cual puede violar el principio de encapsulación. Es preferible mantener el control sobre cómo se accede y se modifica la lista.
2. Modificación no controlada: Si se proporciona un getter para una lista, cualquier código externo puede obtener una referencia directa a la lista y modificarla sin restricciones. Esto puede conducir a un comportamiento inesperado y dificultar el seguimiento de los cambios realizados en la lista.
3. Limitación de flexibilidad: Al exponer una lista directamente a través de un getter, te estás limitando a utilizar una lista como implementación interna. En el futuro, podrías querer cambiar la implementación de la lista a otra estructura de datos, como un conjunto (set) o un mapa (map). Si se han utilizado getters directos en el código, tendrías que actualizar todas las referencias a esos getters, lo cual puede ser tedioso y propenso a errores.

Solución:

En lugar de exponer directamente la lista a través de un getter, una práctica recomendada es proporcionar métodos específicos en tu clase para realizar las operaciones deseadas sobre la lista, como agregar elementos, eliminar elementos, etc. Esto te permite mantener un mejor control sobre la lista y su uso, al tiempo que evita los problemas mencionados anteriormente.

En resumen, evitar los getters de listas como atributos ayuda a mantener un mejor control sobre los datos y preserva el principio de encapsulación en la programación orientada a objetos.

Ejemplo de lo que no se debe hacer:

```
public class ClaseIncorrecta {
    private List<String> lista;

    public List<String> getLista() {
        return lista;
    }
}
```

En este ejemplo, se proporciona un getter directo para la lista. Esto permite que el código externo acceda a la lista y realice modificaciones sin restricciones. Esto puede conducir a problemas de encapsulación y dificultar el seguimiento de los cambios realizados en la lista.

Ejemplo de lo que se debe hacer:

```
public class ClaseCorrecta {
    private List<String> lista;

    public void agregarElemento(String elemento) {
        lista.add(elemento);
    }

    public void eliminarElemento(String elemento) {
        lista.remove(elemento);
    }

    public boolean contieneElemento(String elemento) {
        return lista.contains(elemento);
    }
}
```

En este ejemplo, en lugar de proporcionar un getter directo para la lista, se definen métodos específicos que permiten realizar operaciones sobre la lista de forma controlada. En este caso, se han definido métodos para agregar elementos, eliminar elementos y verificar si la lista contiene un elemento en particular. Estos métodos encapsulan la funcionalidad relacionada con la lista y permiten mantener un mejor control sobre cómo se accede y se modifica la lista.

Al utilizar este enfoque, el código externo no tiene acceso directo a la lista y debe utilizar los métodos proporcionados por la clase para interactuar con ella. Esto ayuda a mantener la coherencia y el control sobre los datos, evitando problemas de encapsulación y facilitando la modificación de la implementación interna de la lista en el futuro si es necesario.

Recuerda que estos ejemplos son solo ilustrativos y que la elección de cómo acceder y modificar los atributos depende del diseño específico de tu aplicación y los requisitos del proyecto.