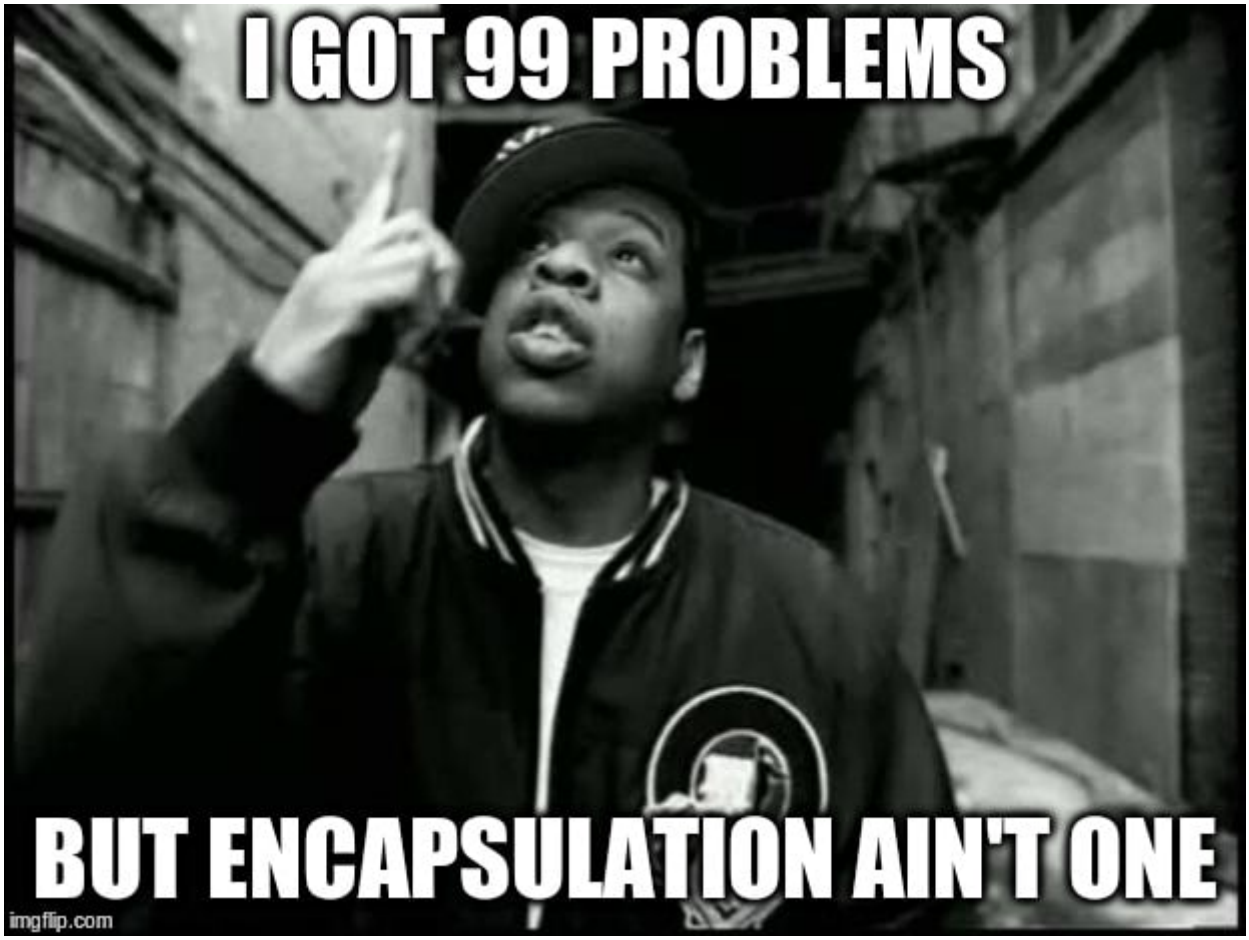
# Deelvraag: Hoe complex wordt de code?

The command pattern encapsulates a request as an object thereby letting you parameterize other objects with different requests que or log requests and support undoable operations.




Het klinkt ingewikkeld, maar het is eenvoudig te implementeren.
De commandos zijn een klasse-object, deze kunnen aangepast of uitgebreid worden door een ander object.

Het Command pattern leent zich ervoor om uit te breiden. Dit kan eenvoudig door een nieuwe methode toe te voegen aan de ontvanger en een nieuwe Command implementatie creeeren zonder de code van de Client aan te passen.

Nadeel van het command pattern is, is dat er veel code bij komt dat verwarrend kan werken doordat er meer methodes toegevoegd worden en veel kleine klassen aangemaakt worden.

## Samengevat:

- Het Command object is de kern van het command design pattern dat het contract definieerd voor de implementatie
- De Receiver(ontvanger) implementatie staat los van het command implementatie

- De Invoker() klasse stuurt alleen het verzoek van de Client naar het command object
- De Client is verantwoordelijk voor het initializeren van het geschikte command en ontvanger objecten en vervolgens deze aan elkaar te koppelen
- De Client is ook verantwoordelijk voor het instantieeren van het Invoker-object en de assosiatie van het command object en het uitvoeren van de execute methode

Receiver Interface:

```
public interface FileSystemReceiver{
    void openFile();
    void writeFile();
    void closeFile();
}
```

Receiver klassen:

```
class UnixFileSystemReceiver implements FileSystemReceiver

    @Override
    public void openFile(){
        System.out.println("Opening file in unix OS")
    }

    @Override
```

## Bronnen

Observation