1 SOLID

2 SRP

Single Responsebility Principle (SRP) hanlder om ikke at lægge mere end én egenskab i hver klasse. Det vil sige at hver klasse skal kun have en opgave (kan fortolkes)

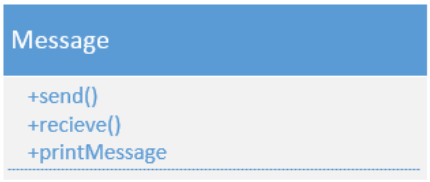


Figure 1: Klasse der ikke overholder SRP

Figur 1 herover viser en Messageklasse som står for kommunikationen i en chat. Klassen kan også skrive det ud den modtager fra vedkommende der chattes med. Dette er et brud på SRP, da det giver klasse mere end en funktion Hertil vil printfunktionen lægges ud i sin egen klasse som det er gjort på figur 2

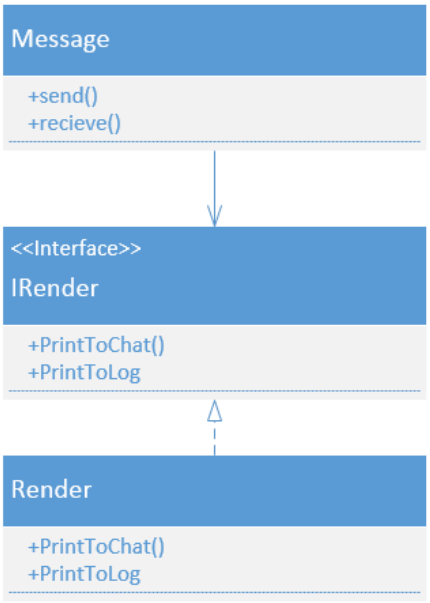


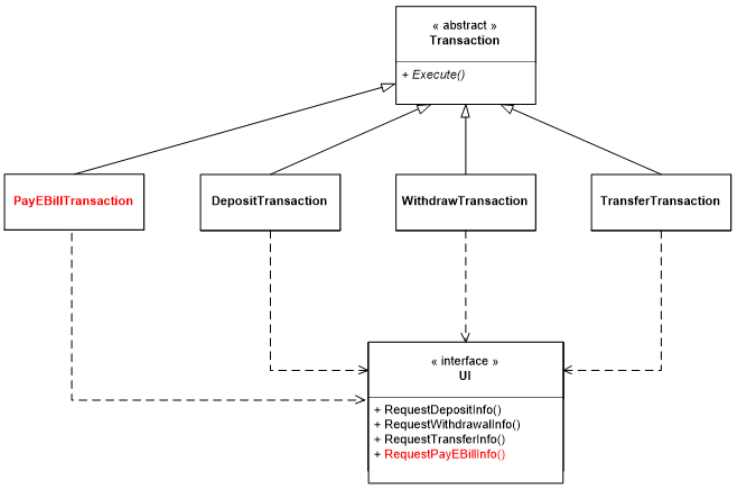
Figure 2: Klasse der overholder SRP

Fordelen ved at separere opgaverne er at vi ikke risikerer at få overfyldte klasser, men også at ændringer i en klasse ikke påvirker for meget i systemet. Dette har også den fordel at andre klasser nu kan anvende Render klassen. **Fordele på listeform:**

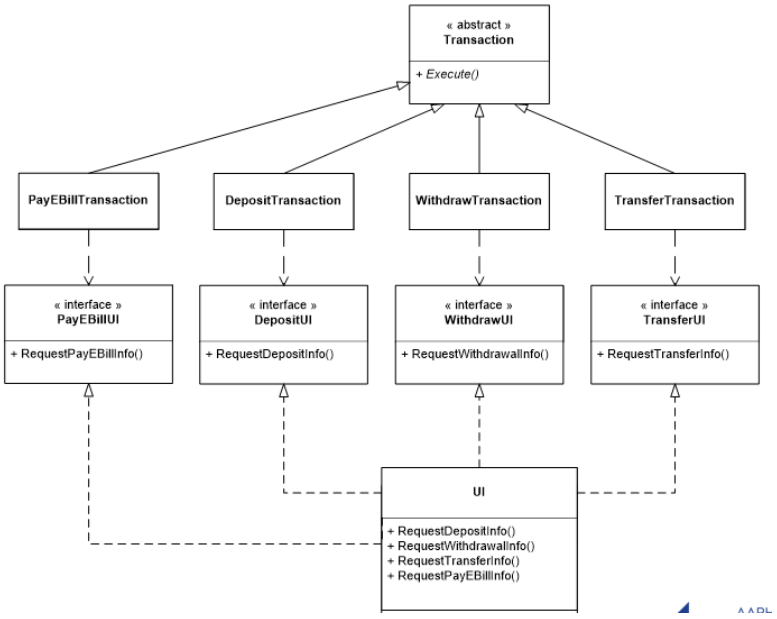
- Mere læselig kode
- Mere maintaineble
- Mere robust kode

3 ISP

Interface segregation principle siger: **"Forskellige typer af klienter bør ikke være afhængig af samme interface. Men bør være afhængig af flere forskellige interfaces til samme *fat class*"**.
Problemet er, at vi ikke ønsker, at klasser skal have adgang til metoder, som de ikke bruger.



Eksemplet ovenfor viser problemet. Her er flere klasser afhængig af samme interface. Og kan dermed kalde flere metoder end de har brug for.
Skal systemet udvides med PayEBillTransaction som på billedet, tilføjes endnu flere funktioner til interfacet, som de andre klasser ikke har brug for, endvidere skal systemet recompiles og re-deployes.
Løsningen er **interface segregation**. Der skal implementeres interface til hver klasse, som så benytter klassen UI. SE BILLEDE NEDENFOR.



Fordelene ved ISP er:

- Lavere kobling - højere maintainability
- Større samhørighed - mere holdbar software.

4 DIP

Dependency Inversion Principal (DIP) handler om at klasserne højere i hierarkiet ikke må være afhængige af de nedre klasser. Eksemplet herunder viser et vandflow-reguleringssystem som har en flowmåler og en ventil. WaterFlow skal her vide hvor mange grader ventilen (Valve) skal drejes, hvilket gør vi her dependency inversion. **"Højniveau klasser må aldrig afhænge af de nedre, men altid af et interace"**

"Abstraktioner bør aldrig afhænge af detaljer. Detlajer bør afhænge af abstraktioner."

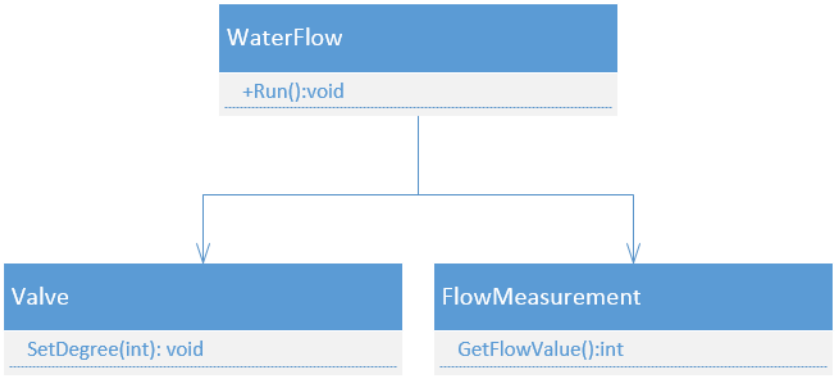
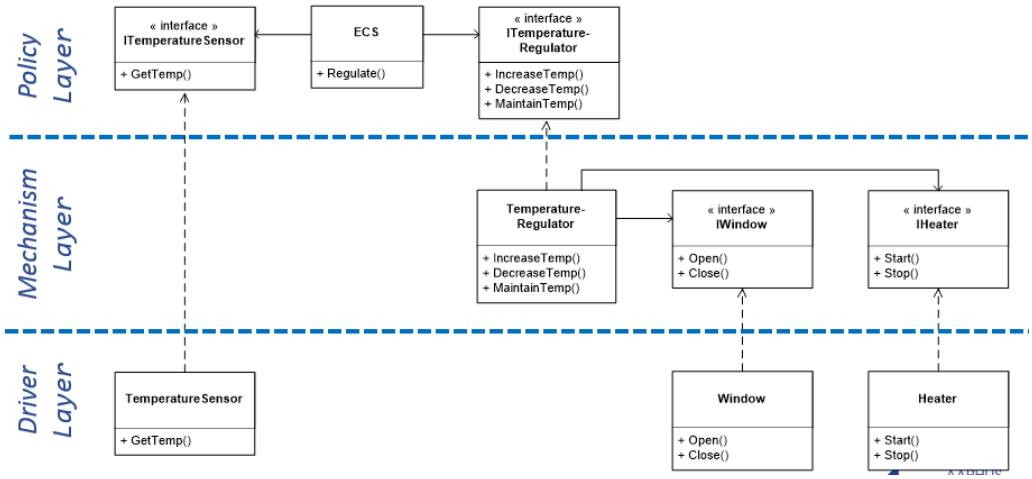


Figure 3: Forkert dependency



I Eksemplet herunder har vi fået løst vores Dependency inversion problem. Der er lagt interfaces ind og WatherFlow skal ikke længere vide hvormeget Valve skal drejes, kun om den skal åbnes eller lukkes.

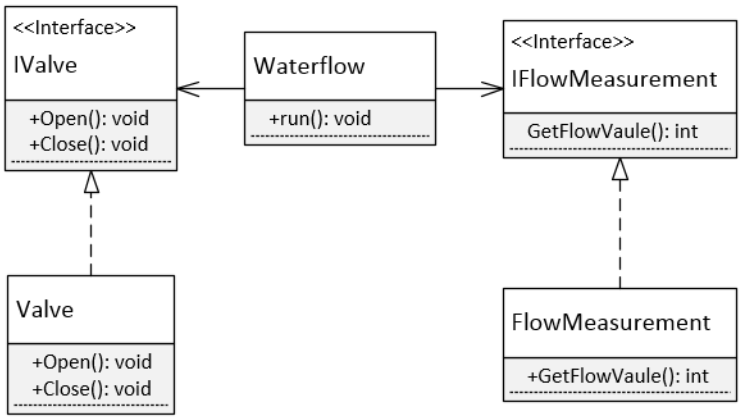


Figure 4: Korrekt Dependency

5 Fordele og ulemper

5.1 SRP

- Hvis man vil overholde SRP kan man nemt komme ud for mange klasser. Det kan give unødvendig kompleksitet, da mange klasser skal til at snakke sammen.
- Desuden bevæger man sig typisk også væk fra real-world forståelsen af et objekt. I virkeligheden vil et objekt have mange ansvar.
- Og det går lidt imod objekt-orienteret programmering, hvor man prøver at modellere verdenen og reducere det gap der er mellem verden og koden.

5.2 ISP

- Som vi så kan det give mange klasser og igen medføre unødvendig kompleksitet.
- Men det er smart at vi kun udstiller de funktioner der er behov for, og det kan også give mindre forvirring internt i koden, hvis man kun udstiller de funktioner som man mener andre programmører på sit team bør kalde.

5.3 DIP

- Nemt at udskifte i da både høj og lav moduler afhænger af abstraktioner ikke hinanden.
- Hvis det aldrig aldrig kunne tænkes at vindue-motoren eller heateren skulle være en anden, så var det unødvendig kompleksitet.