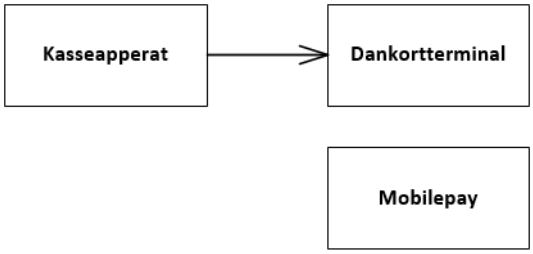
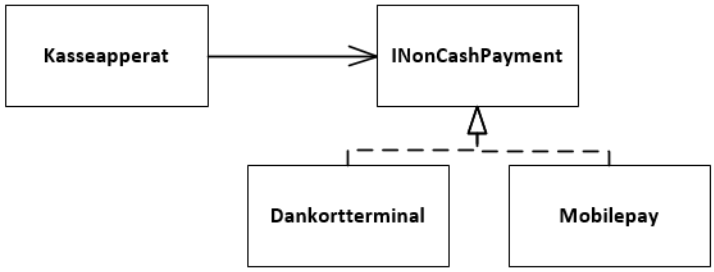
1 Open-close princple

Open-close princippet siger: **"Software entiteter skal være åbne for udvidelser, men lukket for modifikationer"**.
Det betyder, at det skal være muligt at udvide ens software, men ikke modificere eksisterende kode, når/hvis kravene til koden ændres.



Ovenfor er et eksempel, hvor kasseapparatet anvender dankort, men der opstår et problem, når mobilepay bliver en mulighed.
Løsningen er at indsætte interface for NonCashPayments. På den måde kan der udvides i fremtiden, hvis der kommer endnu en betalingsmulighed fx. bitcoins.



Fordele ved OCP:

- Flexibility
- Reuseability
- Maintainability

2 Lisskov substitution princple

"In a computer program, if S is a subtype of T, then objects of type T may be replaced with objects of type S without altering any of the desirable properties of that program"

LSP snakker om at substituere en superklasser med en af denne subklasser. Dette kræver dog at subklassen har samme adfærd som superklassen, ellers risikerer vi fejl når hvadend der anvender dette forventer en anden adfærd.

Et eksempel kan være Ellipse og cirkel. En cirkel er en udgave af en ellipse og bør derfor kunne substistueres. Hvis man i en ellipse ændrer den ene akse, ændres den anden også. Den samme funktionalitet kan ikke findes i en cirkel og derfor kan vi ikke substituere disse.

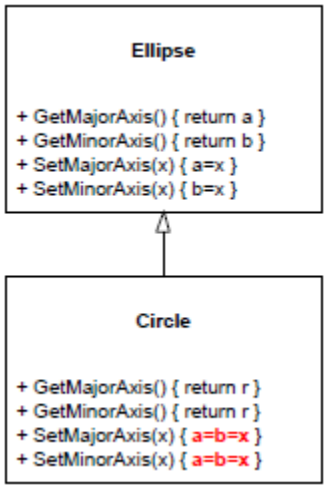


Figure 1: LSP Ellipse

2.1 præ- og postkonditioner

Der er nogle krave for subklassernes prø- og postkonditioner som skal være overholdt hvis disse skal substitueres med deres superklasser:
prækondition En prækondition skal være lig eller svagere suberklassens prækondition.
postkondition En postkondition skal være stærkere eller lig superklassens postkondition

3 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."

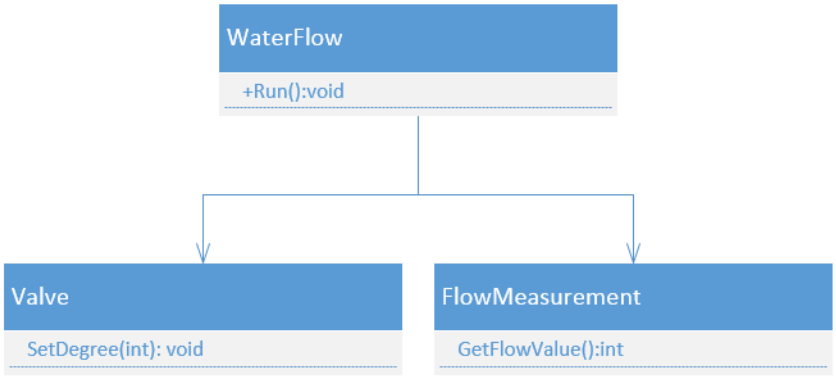
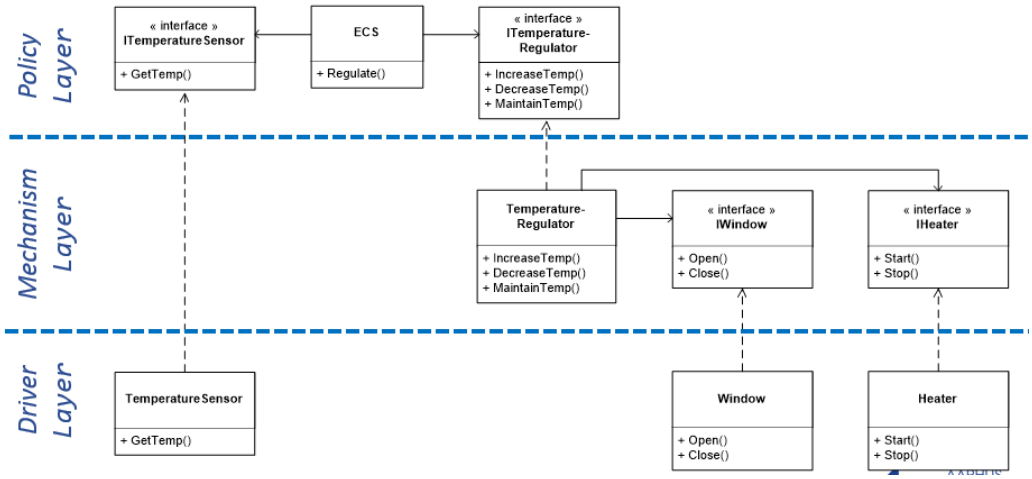


Figure 2: Forkert dependency



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.

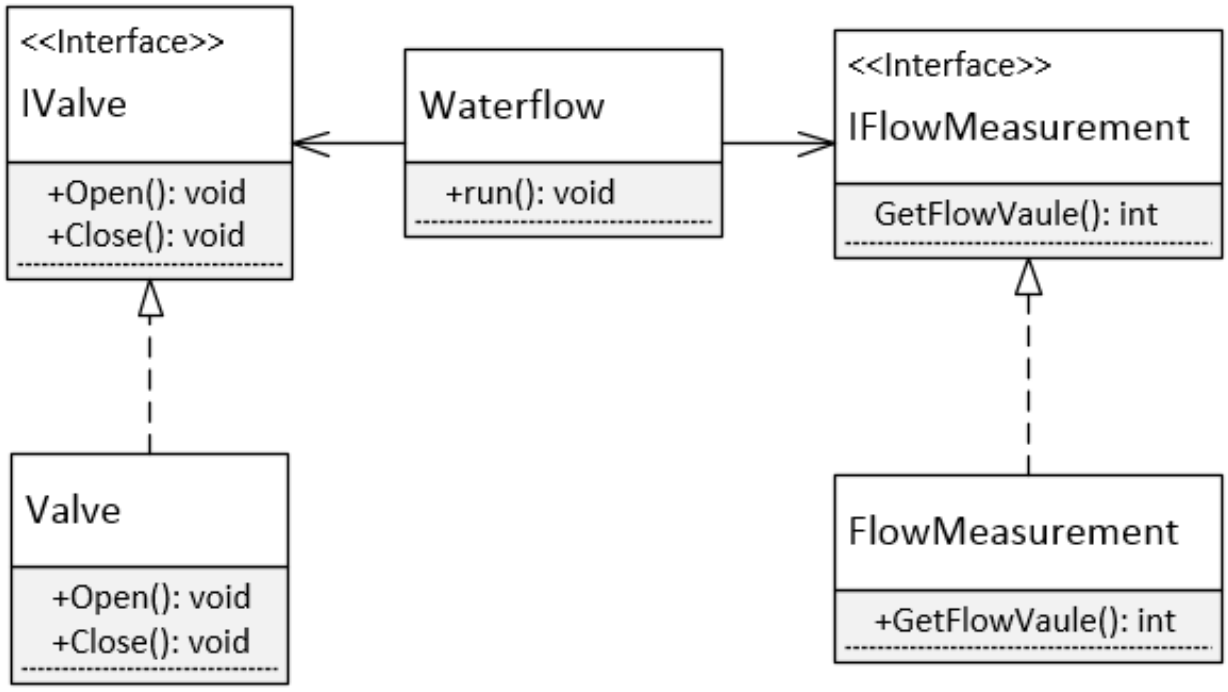


Figure 3: Korrekt Dependency

4 Fordele og ulemper

4.1 OCP

- Nemt at tilføje nyt
- Hvis man tilføjer det alle steder bliver det needless compleksity

4.2 LSP

- +Unit test af superclass vil virke på subclass
- +Giver et godt klasse hiraki

4.3 DIP

- Nemt at udskifte i da både høj og lav moduler afhænger af abstraktioner ikke hinanden.
- -needless complexity(hvis det aldrig kommer til at ændre sig)