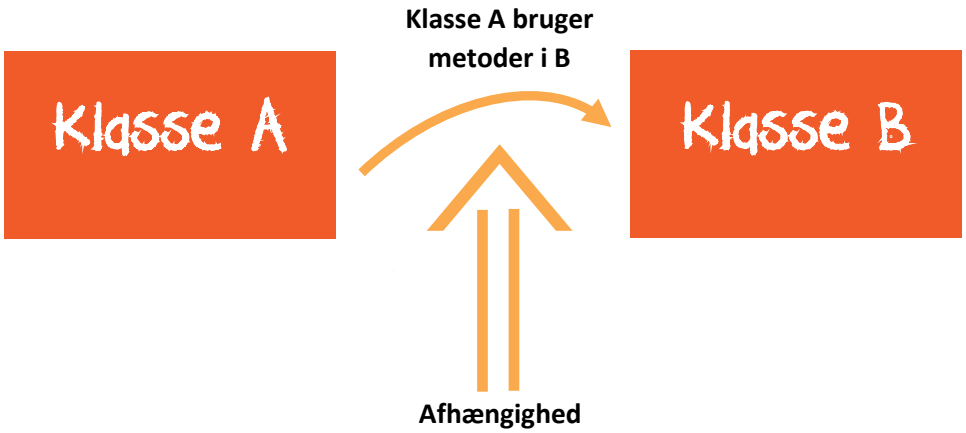
Dependency injection

Dependency injections (DI) er et designmønster, som understøtter SOLID princippet inversion of control. En ”dependency” kan betragtes som en service, som ”injectes” ind i en klient, enten ved brug af setter metode eller via konstruktørkald. Formålet med DI er at ”de-koble ” afhængigheder imellem objekter.

Så inden du læser videre, skal vi måske lige få defineret begrebet afhængighed.

Forstil dig at du har en klasse A. I klasse A gør du brug af metoder i klasse B. A er derfor afhængig af klasse B.



Afhængigheder i programmering er vi ikke særlig glade for. Det skaber en meget tæt kobling klasserne i mellem, hvilket gør det svært at genbruge klasser i andre sammenhæng og det er svært at teste objekternes adfærd i en isoleret tilstand. Endvidere giver afhængigheder også en anden hovedpine. Hvis der bliver ændret på noget i klasse B, kan det få alvorlige konsekvenser for klasse A, hvilket betyder at den dovne programmør får langt mere at lave, end godt er. Først skal klasse B justeres og efterfølgende skal klasse A rettes til og til sidst skal det hele testes igen og så er der måske gået en hel dag.

Lad os prøve at se på et eksempel på hvordan DI kommer til sin fulde ret.

Mennesker kommunikere sammen, på mange forskellige platforme, det kunne f.eks. være via telefon, SMS, Messenger eller e-mail osv. Forskellen i måden vi kommunikerer på, er afhængig af hvilken kommunikationstype, som vælges.

Forestil dig at vi har en klasse message, denne besked skal sendes fra afsender til en modtager. Til dette ville man højest sandsynlig implementere en controller klasse (Postbuddet), som tager sig af at sende beskeden afsted.

Controlleren vil så få et meget stort ansvarsområde (Her brydes der med SRP) i både at skulle implementere alle de forskellige sendMessage metoder for hver kommunikationsform samt afgøre hvilken type besked der er tale om. Nedenstående controller bryder faktisk også med princippet ”Open for extension, but closed for modifications”. Hvad sker der mon, når du af chefen får besked på at sende beskeder via http? Så skal du til at justere i MessageController klassen.

```
public class MessageController
{
    Message m;
    public MessageController(Message message)
    {
        this.m = message;

        if (message is EmailMessage)
        {
            SendMessageViaEmail();
        } else ....
          // Implementering af resten af if
     }

    public void SendMessageViaSMS()
    {
        //Her implementeres alt logikken til at sende beskeden via SMS
    }

    public void SendMessageViaEmail()
    {
        //Her implementeres alt logikken til at sende beskeden via SNMP
    }

    public void SendMessageViaMessenger()
    {
        //Her implementeres alt logikken til at sende beskeden via Messenger
    }
}
```

Ved at gøre brug af dependency injection, kan vi i stedet indkapsle den konkrete implementering i objekter som gør brug af det samme interface og derved opnår vi et mere modulærbart system, hvor der hurtigt kan tilføjes nye transporttyper for kommunikationsformen.

Lad os starte med at definere det overordnede interface Communication.

```
public interface Communication
    {
        void DeliverMessage(Message m);
    }
```

Og efterfølgende en konkret implementering

```
public class EmailTransport : Communication
{
    public void DeliverMessage(Message m)
    {
       //åben smpt forbindelse

       // send besked

       //luk forbindelse
    }
}
```

Controlleren vil herefter, når den initialiseres have et Communication objekt med i f.eks. konstruktørkaldet og det er ligegyldigt, hvilke Communication objekt der sendes med, for selve implementeringen ligger nede i den konkrete implementering.

```
public class MessageController
{
    ICommunication c;
    Message m;
    public MessageController(ICommunication c, Message m)
    {
        this.c = c;
        this.m = m;
     }

    public void SendMessage() {
        c.DeliverMessage(m);
    }
}
```

Man bruger ofte et DI mønster i forbindelse med brugen af persistente lag, altså DAL laget.

Når man snakker om DI, snakker man som regel om 3 forskellige typer.

1. Constructor injection
   DI sker gennem et konstruktørkald som ovenstående eksempel
2. Setter injection
   Klienten skal implementere en setter metode
3. Interface injection
   Det laves et særlig interface som andre klasser kan implementere og så bliver det en slags ”Setter injection”