



UiO : **Institutt for informatikk**
Det matematisk-naturvitenskapelige fakultet

Design og Testbarhet

IN1010 01.04.2020 – Eyvind W. Axelsen



Forelesningen gjennomføres via Zoom

- Dere er i utgangspunktet koblet opp uten lyd og video
- Still spørsmål på chat
- Det blir noen spørsmål fra meg hvor vi prøver Zoom sin «poll-feature»

| Mål for denne forelesningen

Se viktigheten av et godt design

Forstå hva et testbart design er

Kunne skrive enkle programmer på en testbar måte

Kjente feil i programmer

Eller: hvorfor bry seg om design og testbarhet?

Therac 25, 1985-87

- En maskin som ga strålebehandling til pasienter
- En aritmetisk overflyt-feil gjorde at stråler med svært høy energi ble sendt direkte mot pasienter



Hvor stort er et heltall?

Hva er $2\ 147\ 483\ 647 + 1$?

= -2 147 483 648?

Hvilke ting er det viktig å teste for?

Dette henger også sammen med omgivelsene, som f.eks. valg av programmeringsspråk.

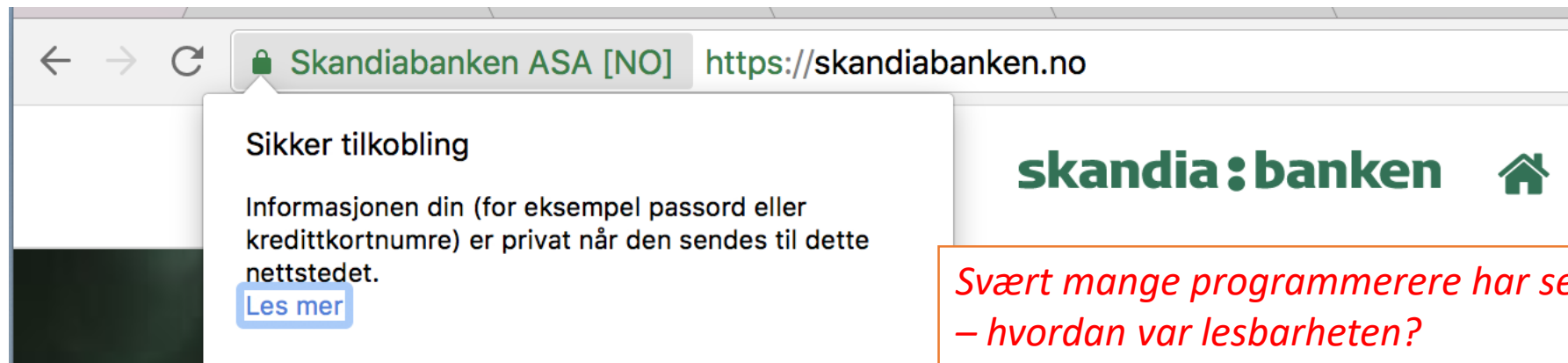
Kjente feil i programmer

Eller: hvorfor bry seg om design og testbarhet?



Heartbleed bug, 2014

- Feil i den svært populære OpenSSL-programvaren (PKI)
- *Én* stakkars programmerer glemte å validere lengden på forespørselen før svaret ble sendt
- Kunne føre til at *hva enn som var i minnet* ble sendt tilbake til en ondsinnet klient, f.eks passord, sertifikater, etc.





Hvor vanlig er det med feil i programmer?

Hva tror du? Hvor mange feil har et program i snitt?

Poll →



Hvor vanlig er det med feil i programmer?

En større studie av over 10 000 000 000(!) kodelinjer fant at det var ca 0,6 – 0,7 feil per 1000 linjer kode.

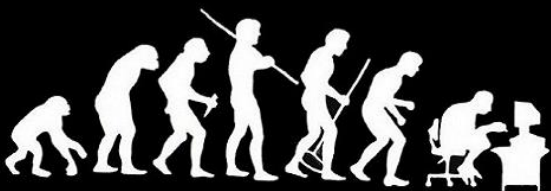
Åpen kildekode er, i gjennomsnitt, bedre enn kommersiell (lukket) kildekode.

Kan god design og testbarhet gjøre at det blir færre feil i programmene våre?

[<http://go.coverity.com/rs/157-LQW-289/images/2014-Coverity-Scan-Report.pdf>]

Hva er programmering?

Tenk litt over saken (vanligvis har vi her en diskusjon med sidemannen, men *desperate times call for desperate measures*, som det heter på nynorsk)



Something, somewhere went terribly wrong



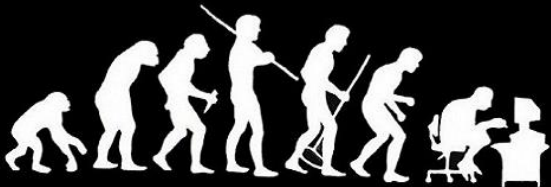
Skriv gjerne på chaten!

Hva er programmering?

Programmering er en prosess hvor man ut fra en beskrivelse av et *beregnbart* problem utvikler et *eksekverbart* **program**.

(Som fortrinnsvis løser problemet.)

Hmm...



Something, somewhere went terribly wrong

Hva er programmering?



Kristen Nygård og Ole-Johan Dahl

«To program is to understand»

- K. Nygård



Kildekode skal typisk leses mye oftere enn den skrives



“Indeed, the ratio of time spent reading versus writing is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write.”

— Robert C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*

- Godt design er viktig!
 - Men, hva er det?
 - Vi ser på noen hovedprinsipper

S
O
L
I
D

5 designprinsipper

S O L I D

- **Single responsibility principle**
En klasse bør bare ha ett ansvarsområde, og dette bør være veldefinert
- **Open/closed principle**
Komponenter bør være åpne for utvidelser, men lukket for modifikasjoner. Eksempel: klasser kontrollerer hva subclasser kan endre (private, protected, final, etc).
- **Liskov substitution principle**
Objekter i et program bør kunne byttes ut med instanser av subtyper uten at dette endrer programmets korrekthet.
- **Interface segregation principle**
Mange små grensesnitt er bedre enn store, monolittiske “super-grensesnitt”
- **Dependency inversion principle**
Komponenter bør avhenge av abstraksjoner, og ikke konkrete implementasjoner

S O L I D

- **Single responsibility principle**
En klasse bør bare ha ett ansvarsområde, og dette bør være veldefinert
- **Open/closed principle**
Komponenter bør være åpne for utvidelser, men lukket for modifikasjoner. Eksempel: klasser kontrollerer hva subclasser kan endre (private, protected, final, etc).
- **Liskov substitution principle**
Objekter i et program bør kunne byttes ut med instanser av subtyper uten at dette endrer programmets korrekthet.
- **Interface segregation principle**
Mange små grensesnitt er bedre enn store, monolittiske “super-grensesnitt”
- **Dependency inversion principle**
Komponenter bør avhenge av abstraksjoner, og ikke konkrete implementasjoner

Designprinsipp:

Single responsibility – ett ansvarsområde

- En klasse bør ha ett veldefinert ansvarsområde
- Når endringer må gjøres i et program, så bør det bare være én grunn til å endre en gitt klasse
- Prinsippet er ment å sikre robusthet og forståelighet
- Tett knyttet til prinsippet om «separation of concerns» → ikke bland alt sammen i en diger smørje!
 - Hvert «concern» er et område av applikasjonen, som bør ha en selvstendig implementasjon

Har denne klassen ett veldefinert ansvarsområde?

```
public class Ansatt {  
    String fornavn;  
    String etternavn;  
    double lønn;  
    String stilling;  
    double stillingsProsent;  
    Ansatt sjef;  
  
    public Ansatt(String fornavn, String etternavn, double lønn, String stilling,  
                  double stillingsProsent, Ansatt sjef) throws IllegalStateException {  
  
        this.fornavn = fornavn;  
        this.etternavn = etternavn;  
        this.lønn = lønn;  
        this.stilling = stilling;  
        this.sjef = sjef;  
  
        if(stillingsProsent > 100)  
            throw new IllegalStateException("Kan ikke jobbe mer enn 100 %!");  
  
        this.stillingsProsent = stillingsProsent;  
    }  
  
    public void justerLønnMed(double beløp) throws IllegalStateException {  
        lønn = lønn + beløp;  
  
        if(lønn > sjef.lønn)  
            throw new IllegalStateException("Ingen kan tjene mer enn sjefen!");  
    }  
}
```

Har denne klassen ett veldefinert ansvarsområde?

```
public class Ansatt {
```

```
    String fornavn;  
    String etternavn;  
    double lønn;  
    String stilling;  
    double stillingsProsent;  
    Ansatt sjef;
```

```
    public Ansatt(String fornavn, String etternavn, double lønn, String stilling,  
                 double stillingsProsent, Ansatt sjef) throws IllegalStateException {
```

```
        this.fornavn = fornavn;  
        this.etternavn = etternavn;  
        this.lønn = lønn;  
        this.stilling = stilling;  
        this.sjef = sjef;
```

```
        if(stillingsProsent > 100)  
            throw new IllegalStateException("Kan ikke jobbe mer enn 100 %!");
```

```
        this.stillingsProsent = stillingsProsent;
```

```
    }
```

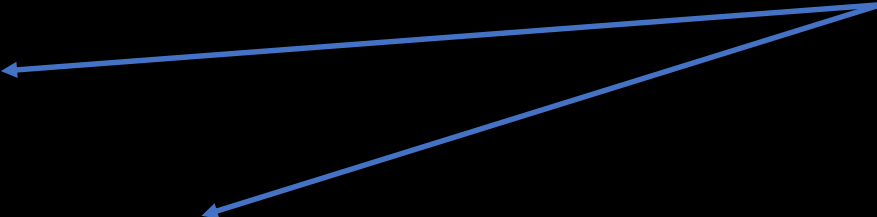
```
    public void justerLønnMed(double beløp) throws IllegalStateException {  
        lønn = lønn + beløp;
```

```
        if(lønn > sjef.lønn)  
            throw new IllegalStateException("Ingen kan tjene mer enn sjefen!");
```

```
    }
```

```
}
```

Sammenblanding av
ansatt- og stillingsdata,
og regler for stillinger



Har denne klassen ett veldefinert ansvarsområde?

```
public class Ansatt {
```

```
    String fornavn;  
    String etternavn;  
    double lønn;  
    String stilling;  
    double stillingsProsent;  
    Ansatt sjef;
```

```
    public Ansatt(String fornavn, String etternavn, double lønn, String stilling,  
                 double stillingsProsent, Ansatt sjef) throws IllegalStateException {
```

```
        this.fornavn = fornavn;  
        this.etternavn = etternavn;  
        this.lønn = lønn;  
        this.stilling = stilling;  
        this.sjef = sjef;
```

```
        if(stillingsProsent > 100)  
            throw new IllegalStateException("Kan ikke jobbe mer enn 100 %!");
```

```
        this.stillingsProsent = stillingsProsent;
```

```
    }
```

```
    public void justerLønnMed(double beløp) throws IllegalStateException {  
        lønn = lønn + beløp;
```

```
        if(lønn > sjef.lønn)  
            throw new IllegalStateException("Ingen kan tjene mer enn sjefen!");
```

```
    }
```

```
}
```

Sammenblanding med
organisasjonskart og
hierarki



Har denne klassen ett veldefinert ansvarsområde?

```
public class Ansatt {
```

```
    String fornavn;  
    String etternavn;  
    double lønn;  
    String stilling;  
    double stillingsProsent;  
    Ansatt sjef;
```

```
    public Ansatt(String fornavn, String etternavn, double lønn, String stilling,  
                 double stillingsProsent, Ansatt sjef) throws IllegalStateException {
```

```
        this.fornavn = fornavn;  
        this.etternavn = etternavn;  
        this.lønn = lønn;  
        this.stilling = stilling;  
        this.sjef = sjef;
```

```
        if(stillingsProsent > 100)  
            throw new IllegalStateException("Kan ikke jobbe mer enn 100 %!");
```

```
        this.stillingsProsent = stillingsProsent;
```

```
    }  
  
    public void justerLønnMed(double beløp) throws IllegalStateException {  
        lønn = lønn + beløp;
```

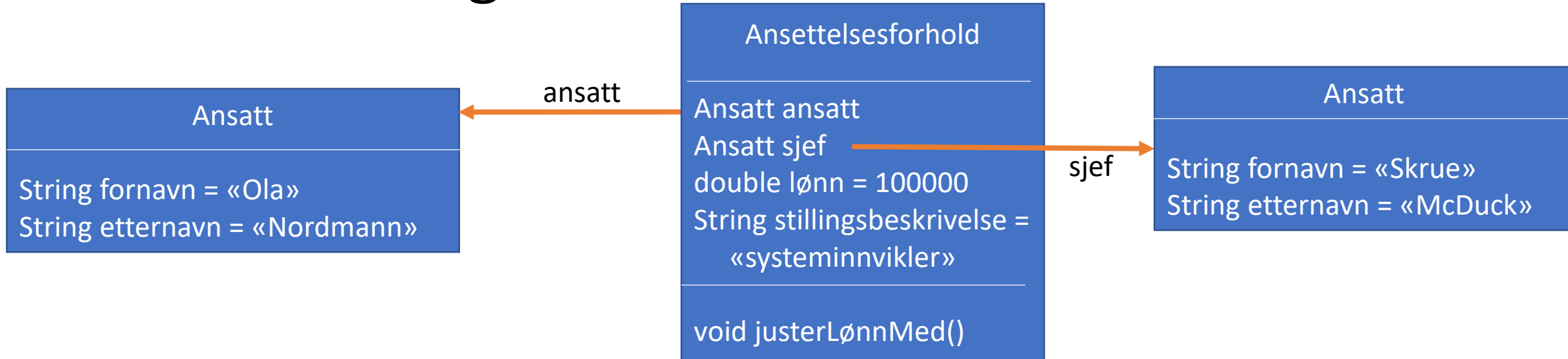
```
        if(lønn > sjef.lønn)  
            throw new IllegalStateException("Ingen kan tjene mer enn sjefen!");
```

```
    }
```

```
}
```

Sammenblanding med
forretningslogikk

Et bedre design?



```
class Ansettelsesforhold {
    String stilling;
    double stillingsProsent, lønn;
    Ansatt ansatt, sjef;

    public Ansettelsesforhold(Ansatt ansatt, double lønn, String stilling, double
        stillingsProsent, Ansatt sjef) throws IllegalStateException {
        this.ansatt = ansatt;
        this.lønn = lønn;
        this.stilling = stilling;
        this.stillingsProsent = stillingsProsent;
        this.sjef = sjef;

        if (stillingsProsent > 100)
            throw new IllegalStateException("Kan ikke jobbe mer enn 100 %!");

        this.stillingsProsent = stillingsProsent;
    }

    public void økLønnMed(double beløp) throws IllegalStateException {
        lønn = lønn + beløp;
        if (lønn > sjef.lønn)
            throw new IllegalStateException("Ingen kan tjene mer enn sjefen!");
    }
}
```

Designprinsipp:

Dependency inversion – omvendte avhengigheter

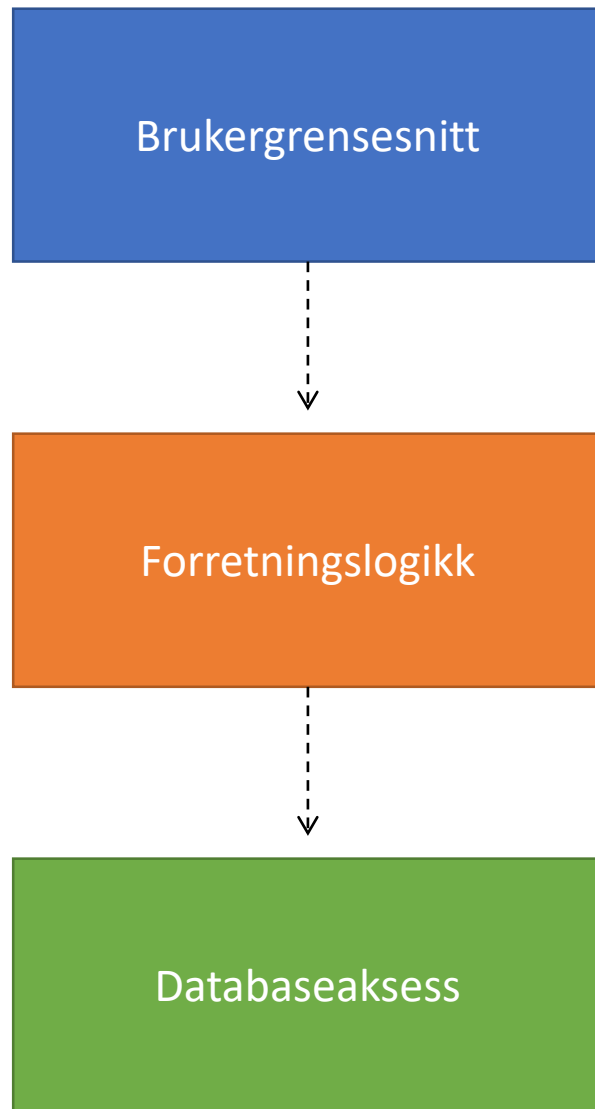
- Moduler på et høyt nivå bør ikke avhenge av moduler på lavere nivåer
- Abstraksjoner bør ikke avhenge av implementasjoner, men implementasjonene avhenger i stedet av abstraksjonene
- Nivået som trenger abstraksjonene eier dem





Hva er et interface ?

- Et interface ligner en abstrakt klasse
- **Alle** metodene i et interface er abstrakte og polymorfe
- En interface inneholder **ingen** variable eller annen datastruktur (men litt annet som vi ikke bruker i IN1010)
- En klasse som arver egenskapene til et interface må selv putte inn kode i alle de abstrakte metodene (og deklarerer passende variable som disse metodene bruker for å gjøre jobben sin).
- En klasse kan arve egenskapene til mange grensesnitt (men bare en klasse)
- Å arve (en samling metoder) = å spille en rolle



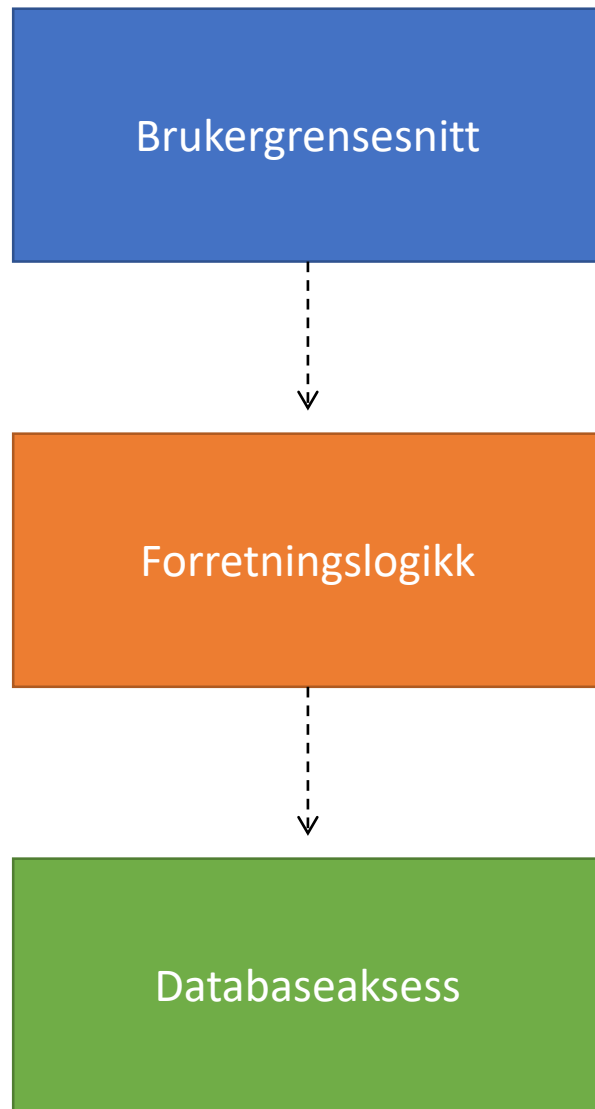
«Tradisjonell» design:

- Brukergrensesnittet er avhengig av forretningslogikk, som igjen er avhengig av et databasenært lag.
- Hver boks kan representere et sett av klasser

Hva er problemet med dette?

- Tette knytninger
- Vanskelig å teste komponenter i isolasjon

↑
Avhengighet

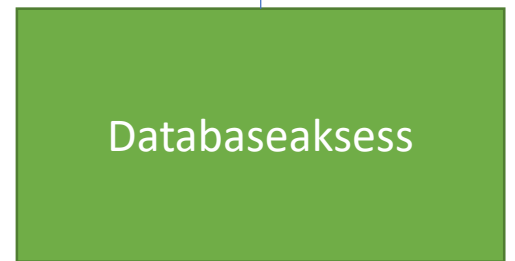
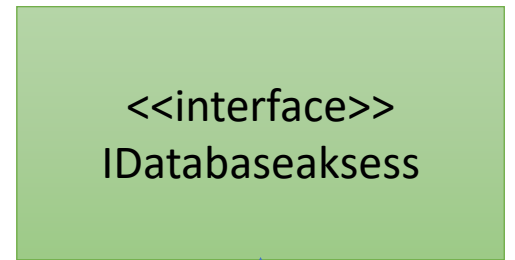
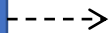
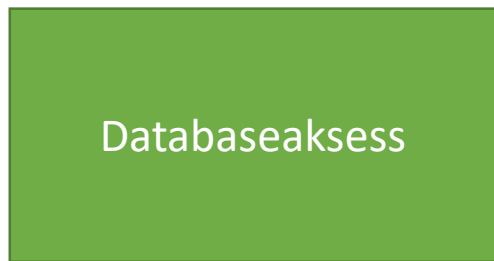
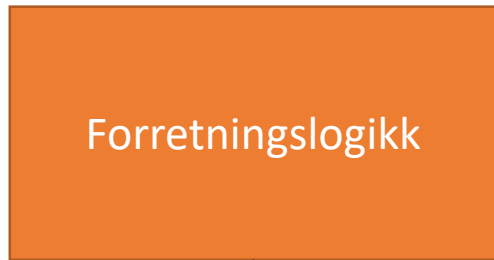


Avhengighet, sier du? Hva betyr det?

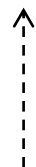
```
class A {  
    B b = new B();  
}  
  
class B {  
    /* ... */  
}
```

A har en avhengighet til B

↑
Avhengighet



Hvert lag eier abstraksjonene det er avhengig av



Avhengighet



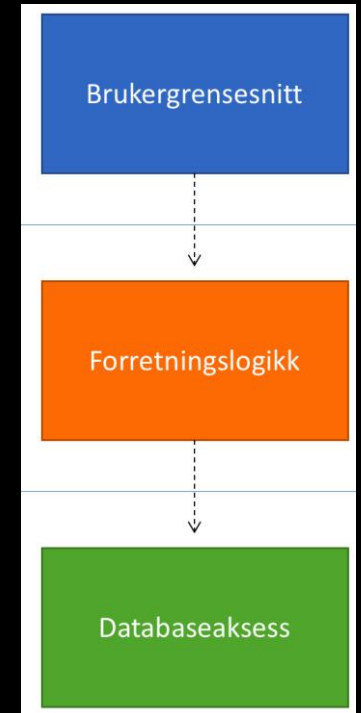
Implementasjon

Uten DI

```
class Brukergrensesnitt {  
    Forretningslogikk forretningslogikk = new Forretningslogikk();  
    // ...  
}
```

```
class Forretningslogikk {  
    Databaseaksess dataaksess = new Databaseaksess();  
    // ...  
}
```

```
class Databaseaksess {  
    // ...  
}
```



```
interface IForretningslogikk { /* ... */ }
```

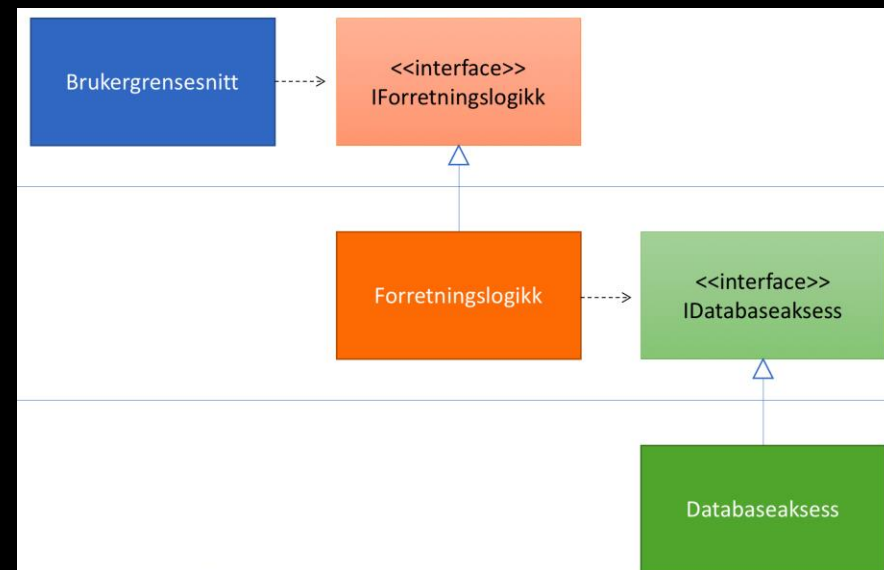
```
class Brukergrensesnitt {  
    public Brukergrensesnitt(IForretningslogikk forretningslogikk) { /* ... */ }  
    // ...  
}
```

```
interface IDatabaseaksess { /* ... */ }
```

```
class Forretningslogikk implements IForretningslogikk {  
    public Forretningslogikk(IDatabaseaksess databaseaksess) { /* .. */ }  
    // ...  
}
```

```
class Databaseaksess implements IDatabaseaksess {  
    // ...  
}
```

Eksplisitt avhengighet til grensesnitt
Kan *byttes ut* f.eks. under testing



Testbarhet – å legge til rette for programmatisk testing

Hvis man skal teste et program, så er første bud at kriteriene det testes for er klare

- Kriteriene for et riktig program må være
 - Utvetydige
 - Kvantitative/målbare
 - Verifiserbare i praksis
- Et testbart program bør være modularisert på en slik måte at hver komponent lar seg teste for seg
 - Enhetstesting – hver komponent testes i isolasjon
 - Ikke så enkelt som det kanskje høres ut som – dette må kodes inn eksplisitt!
 - Integrasjonstesting – samspillet mellom enkeltkomponenter testes

Har du skrevet en test før?

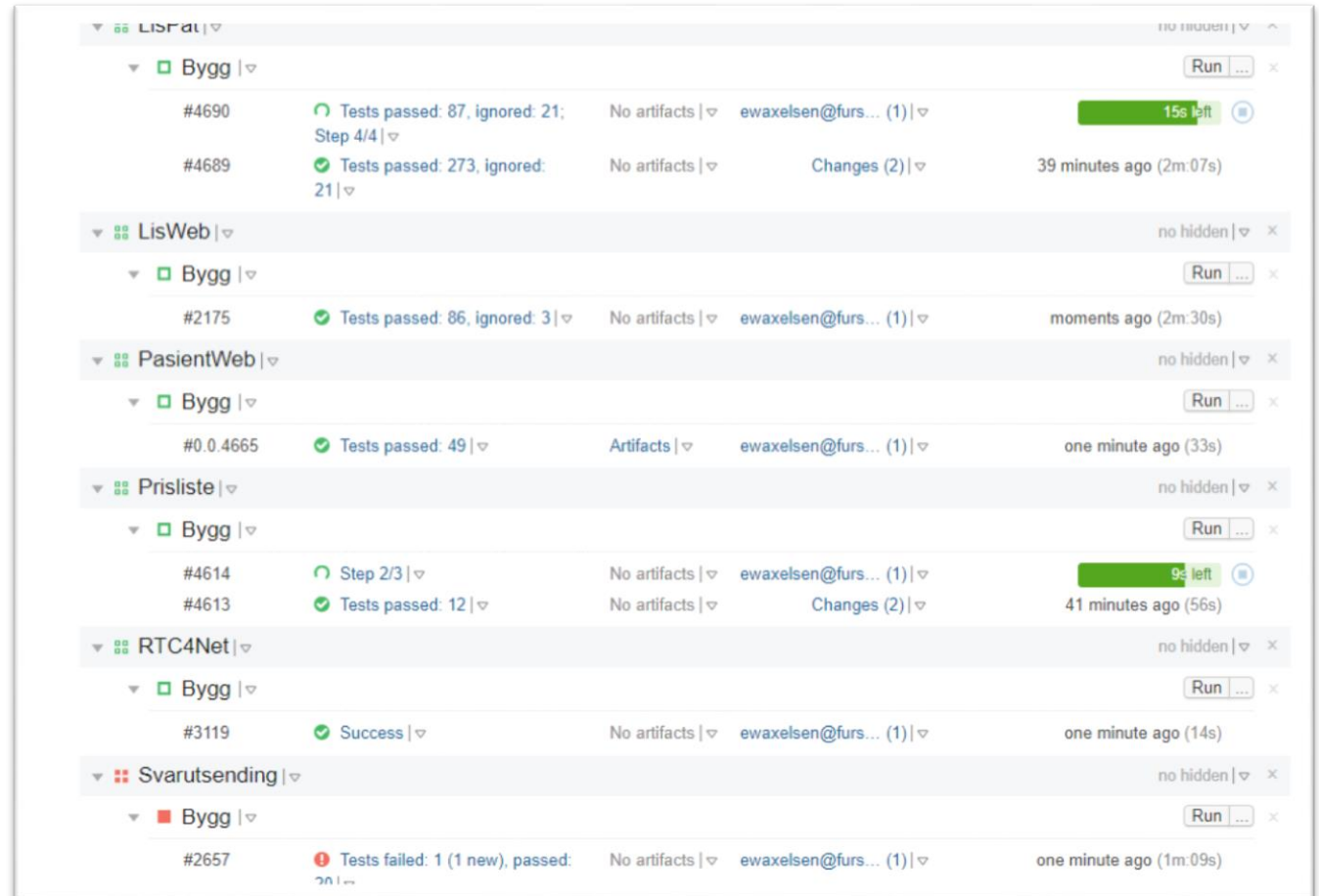
Poll →

Hvordan tester du dine programmer?

- Skriv gjerne inn på chat-en!

CI – Continuous Integration («kontinuerlig integrasjon»)

- Alle utviklere på et prosjekt sjekker inn/«commiter» kode kontinuerlig, f.eks. til git
- Hele prosjektet bygges automatisk ved hver innsjekk
- Tester kjøres automatisk ved hvert bygg for å forhindre regresjoner
- Regresjon: kode som tidligere fungerte har blitt ødelagt



The screenshot displays a CI/CD pipeline dashboard with the following details:

- Build Jobs:** Multiple jobs are listed, each with a unique ID (e.g., #4690, #4689, #2175, #0.0.4665, #4614, #4613, #3119, #2657).
- Status:** Jobs are marked as 'Success' (green checkmark) or 'Step 2/3' (green circle with checkmark). One job (#2657) is marked as 'Tests failed: 1 (1 new), passed: 20' (red exclamation mark).
- Test Results:** Test counts are provided for several jobs, such as 'Tests passed: 87, ignored: 21' for job #4690 and 'Tests passed: 273, ignored: 21' for job #4689.
- Artifacts:** Some jobs have 'No artifacts' or 'Artifacts' listed.
- Changes:** Jobs like #4689 and #4613 show 'Changes (2)'. Job #4614 shows 'Step 2/3'.
- Timing:** Completion times are shown, such as '15s left' for job #4690, '39 minutes ago (2m:07s)' for job #4689, and 'one minute ago (14s)' for job #3119.
- Module Structure:** The dashboard is organized into sections for different modules: LisFat, LisWeb, PasientWeb, Prislister, RTC4Net, and Svarutsending.

Et eksempel: Studentlisteprinter

Krav til programmet:

- Skal definere et konsept «Student», med fornavn, etternavn og brukernavn
- Studentene skal skrives ut som en liste på papir, sortert etter brukernavn
- Formen på utskriften skal være slik:
«Ola Olsen (olao)»
- Programmet skal være testbart

```
class Student
```

```
String fornavn  
String etternavn  
String brukernavn
```

```
String getFornavn()  
String getEtternavn()  
String getBrukernavn()
```

```
class Program {
    public static void main(String[] args) throws PrintException, IOException {
        List<Student> studenter = new ArrayList<Student>();
        studenter.add(new Student("Dag", "Langmyhr", "dag"));
        studenter.add(new Student("Stein", "Gjessing", "steing"));
        studenter.add(new Student("Eyvind", "Axelsen", "eyvinda"));

        StudentPrinter studprint = new StudentPrinter();
        studprint.print(studenter);
    }
}

class StudentPrinter {
    public void print(List<Student> studenter) throws PrintException, IOException {

        for (Student student : studenter) {
            String data = student.getFornavn() + student.getEtternavn() + student.getBrukernavn();

            try (InputStream is = new ByteArrayInputStream(data.getBytes())) {
                DocFlavor flavor = DocFlavor.INPUT_STREAM.AUTOSENSE;
                PrintService service = PrintServiceLookup.lookupDefaultPrintService();
                // Create the print job
                DocPrintJob job = service.createPrintJob();
                Doc doc = new SimpleDoc(is, flavor, null);
                // Monitor print job events; for the implementation of PrintJobWatcher,
                PrintJobWatcher pjDone = new PrintJobWatcher(job);
                // Print it
                job.print(doc, null);
                // Wait for the print job to be done
                pjDone.waitForDone();
            }
        }
    }
}
```

```

class Program {
    public static void main(String[] args) throws PrintException, IOException {
        List<Student> studenter = new ArrayList<Student>();
        studenter.add(new Student("Dag", "Langmyhr", "dag"));
        studenter.add(new Student("Stein", "Gjessing", "steing"));
        studenter.add(new Student("Eyvind", "Axelsen", "eyvinda"));

        StudentPrinter studprint = new StudentPrinter();
        studprint.print(studenter);
    }
}

class StudentPrinter {
    public void print(List<Student> studenter) throws PrintException, IOException {

        for (Student student : studenter) {
            String data = student.getFornavn() + student.getEtternavn() + student.getBrukernavn();

            try (InputStream is = new ByteArrayInputStream(data.getBytes())) {
                DocFlavor flavor = DocFlavor.INPUT_STREAM.AUTOSENSE;
                PrintService service = PrintServiceLookup.lookupDefaultPrintService();
                // Create the print job
                DocPrintJob job = service.createPrintJob();
                Doc doc = new SimpleDoc(is, flavor, null);
                // Monitor print job events; for the implement
                PrintJobWatcher pjDone = new PrintJobWatcher
                // Print it
                job.print(doc, null);
                // Wait for the print job to be done
                pjDone.waitForDone();
            }
        }
    }
}

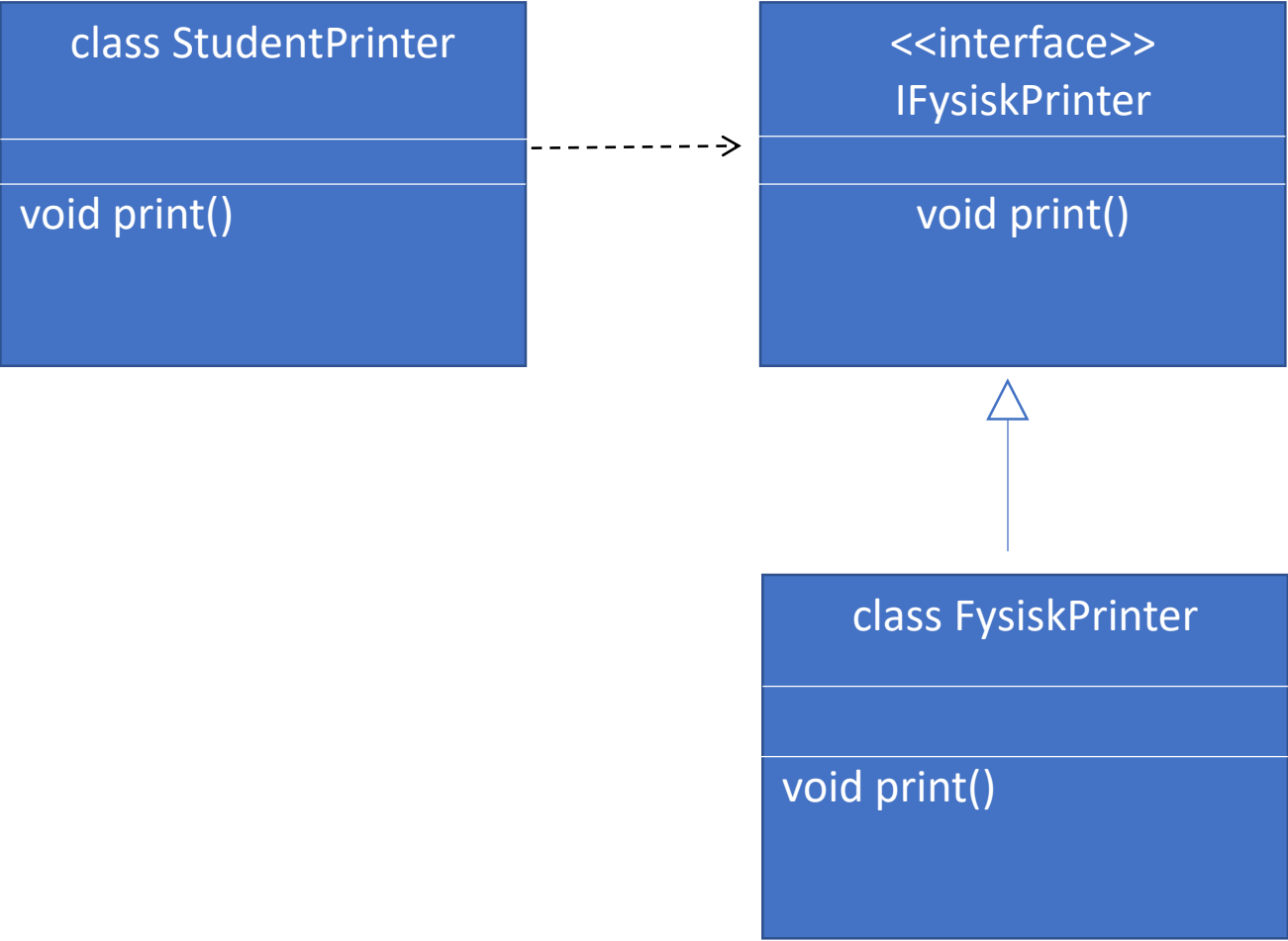
```

- Skal definere et konsept «Student», med fornavn, etternavn og brukernavn
- Studentene skal skrives ut som en liste på papir, sortert etter brukernavn
- Formen på utskriften skal være slik:
«Ola Olsen (olao)»
- Programmet skal være testbart
- Er det noe her du kunne tenke deg å endre på? Kommenter gjerne på chat!

Observasjoner

- StudentPrinter-klassen gjør (minst) to ting:
 - Henter ut data som skal printes fra Student-objekter
 - Gjør fysisk utskrift
 - Vi prøver oss på å skille disse fra hverandre!
 - Single responsibility principle
- Det er dumt at det fyker papir ut på printeren for hver gang man kjører en (automatisert) test
 - Vi trekker ut fysisk print som et eget grensesnitt
 - Benytter dependency inversion for å sette opp avhengighetene





```
interface IFysiskPrinter {  
    public void print(String data) throws PrintException, IOException ;  
}
```

```
class StudentPrinter {  
    IFysiskPrinter fysiskPrinter;
```

```
    public StudentPrinter(IFysiskPrinter fysiskPrinter) {  
        this.fysiskPrinter = fysiskPrinter;  
    }
```

```
    public void print(List<Student> studenter) throws PrintException, IOException {
```

```
        // sortere på brukernavn:
```

```
        studenter.sort((s1, s2) -> s1.getBrukernavn().compareTo(s2.getBrukernavn()));
```

```
        // løp gjennom alle studenter, og skriv ut
```

```
        for (Student student : studenter) {
```

```
            String data = student.getFornavn() + student.getEtternavn() +  
                student.getBrukernavn();
```


```
            fysiskPrinter.print(data);
```

```
        }
```

```
    }
```

```
}
```

Eksplisitt avhengighet til
abstraksjon - dependency
inversion



3 A-er: Arrange, Act, Assert

Oppsett for enhetstester

- **Arrange**: gjør alt oppsett for testen
- **Act**: utfør selve handlingen som skal testes
- **Assert**: sjekk at vi fikk forventet resultat

```
class StudentPrinterTester {
```

```
public void print_KaltMed3Studenter_SkriverUtStudenterIRiktigRekkefølge() throws Exception {
```

```
    // arrange
```

```
    List<String> utskrift = new ArrayList<String>();
```

```
    StudentPrinter eut = new StudentPrinter(
```

```
        new IFysiskPrinter() {
```

```
            public void print(String data){
```

```
                utskrift.add(data);
```

```
            }
```

```
        }
```

```
    );
```

```
    List<Student> studenter = new ArrayList<Student>();
```

```
    studenter.add(new Student("Dag", "Langmyhr", "dag"));
```

```
    studenter.add(new Student("Stein", "Gjessing", "steing"));
```

```
    studenter.add(new Student("Eyvind", "Axelsen", "eyvinda"));
```

```
    // act
```

```
    eut.print(studenter);
```

```
    // assert
```

```
    if(!utskrift.get(0).equals("Dag Langmyhr (dag)")) throw new RuntimeException("Feil");
```

```
    if(!utskrift.get(1).equals("Eyvind Axelsen (eyvinda)")) throw new RuntimeException("Feil");
```

```
    if(!utskrift.get(2).equals("Stein Gjessing (steing)")) throw new RuntimeException("Feil");
```

```
    }
```

```
}
```

```
// Se foil 45 for eksempel som ikke bruker inline interface
```

```
class DummyPrinter implements IFysiskPrinter {
```

```
    public List<String> utskrift = new ArrayList<String>();
```

```
    public void print() {
```

```
        utskrift.add(data);
```

```
    }
```

```
}
```

```
class StudentPrinterTester {
```

```
public void print_KaltMed3Studenter_SkriverUtStudenterIRiktigRekkefølge() throws Exception {
```

```
    // arrange
```

```
    List<String> utskrift = new ArrayList<String>();
```

```
    StudentPrinter eut = new StudentPrinter(
```

```
        new IFysiskPrinter() {
```

```
            public void print(String data){
```

```
                utskrift.add(data);
```

```
            }
```

```
        }
```

```
    );
```

```
    List<Student> studenter = new ArrayList<Student>();
```

```
    studenter.add(new Student("Dag", "Langmyhr", "dag"));
```

```
    studenter.add(new Student("Stein", "Gjessing", "steing"));
```

```
    studenter.add(new Student("Eyvind", "Axelsen", "eyvinda"));
```

```
    // act
```

```
    eut.print(studenter);
```

```
    // assert
```

```
    if(!utskrift.get(0).equals("Dag Langmyhr (dag)")) throw new RuntimeException("Feil");
```

```
    if(!utskrift.get(1).equals("Eyvind Axelsen (eyvinda)")) throw new RuntimeException("Feil");
```

```
    if(!utskrift.get(2).equals("Stein Gjessing (steing)")) throw new RuntimeException("Feil");
```

```
    }
```

```
}
```

DI lar oss bytte ut den fysiske
printeren med en dummy
testversjon

Assert-delen sjekker at
utskriften faktisk blir som
spesifisert

```
class StudentPrinterTester {
```

```
    @Test void print_KaltMed3Studenter_SkriverUtStudenterIRiktigRekkefølge() {
```

```
        // arrange
```

```
        List<String> utskrift = new ArrayList<String>();
```

```
        StudentPrinter eut = new StudentPrinter(
```

```
            new IFysiskPrinter() {
```

```
                public void print(String data){
```

```
                    utskrift.add(data);
```

```
                }
```

```
            }
```

```
        );
```

```
        List<Student> studenter = new ArrayList<Student>();
```

```
        studenter.add(new Student("Dag", "Langmyhr", "dag"));
```

```
        studenter.add(new Student("Stein", "Gjessing", "steing"));
```

```
        studenter.add(new Student("Eyvind", "Axelsen", "eyvinda"));
```

```
        // act
```

```
        eut.print(studenter);
```

```
        // assert
```

```
        assertEquals("Dag Langmyhr (dag)", utskrift.get(0));
```

```
        assertEquals("Eyvind Axelsen (eyvinda)", utskrift.get(1));
```

```
        assertEquals("Stein Gjessing (steing)", utskrift.get(2));
```

```
    }
```

```
}
```

```
import static org.junit.jupiter.api.Assertions.*;
```

```
import org.junit.jupiter.api.Test;
```

```
class StudentPrinterTester {
```

```
    @Test void print_KaltMed3Studenter_SkriverUtStudenterIRiktigRekkefølge() {
```

```
        // arrange
```

```
        List<String> utskrift = new ArrayList<String>();
```

```
        StudentPrinter eut = new StudentPrinter(
```

```
            new IFysiskPrinter() {
```

```
                public void print(String data){
```

```
                    utskrift.add(data);
```

```
                }
```

```
            }
```

```
        );
```

```
        List<Student> studenter = new ArrayList<Student>();
```

```
        studenter.add(new Student("Dag", "Langmyhr", "dag"));
```

```
        studenter.add(new Student("Stein", "Gjessing", "steing"));
```

```
        studenter.add(new Student("Eyvind", "Axelsen", "eyvinda"));
```

```
        // act
```

```
        eut.print(studenter);
```

```
        // assert
```

```
        assertEquals("Dag Langmyhr (dag)", utskrift.get(0));
```

```
        assertEquals("Eyvind Axelsen (eyvinda)", utskrift.get(1));
```

```
        assertEquals("Stein Gjessing (steing)", utskrift.get(2));
```

```
    }
```

```
}
```

Coverage Report - All Packages

Package /	# Classes	Line Coverage	Branch Coverage
All Packages	221	84% 2970/3513	81% 859/1060
junit.extensions	6	82% 52/63	87% 7/8
junit.framework	17	76% 399/525	90% 139/154
junit.runner	3	49% 77/155	41% 23/56
junit.textui	2	76% 99/130	76% 23/30
org.junit	14	85% 196/230	75% 68/90
org.junit.experimental	2	91% 21/23	83% 5/6
org.junit.experimental.categories	5	100% 67/67	100% 44/44
org.junit.experimental.max	8	85% 92/108	86% 26/30
org.junit.experimental.results	6	92% 37/40	87% 7/8
org.junit.experimental.runners	1	100% 2/2	N/A N/A
org.junit.experimental.theories	14	96% 119/123	88% 37/42
org.junit.experimental.theories.internal	5	88% 98/111	92% 39/42
org.junit.experimental.theories.suppliers	2	100% 7/7	100% 2/2
org.junit.internal	11	94% 149/157	94% 53/56
org.junit.internal.builders	8	98% 57/58	92% 13/14
org.junit.internal.matchers	4	75% 40/53	0% 0/18
org.junit.internal.requests	3	96% 27/28	100% 2/2
org.junit.internal.runners	18	73% 306/415	63% 82/130
org.junit.internal.runners.model	3	100% 26/26	100% 4/4
org.junit.internal.runners.rules	1	100% 35/35	100% 20/20
org.junit.internal.runners.statements	7	97% 92/94	100% 14/14
org.junit.matchers	1	9% 1/11	N/A N/A
org.junit.rules	20	89% 203/226	96% 31/32
org.junit.runner	12	93% 150/161	88% 30/34
org.junit.runner.manipulation	9	85% 36/42	77% 14/18
org.junit.runner.notification	12	100% 98/98	100% 8/8
org.junit.runners	16	98% 321/327	96% 95/98
org.junit.runners.model	11	82% 163/198	73% 73/100

Report generated by [Cobertura](#) 1.9.4.1 on 12/22/12 2:25 PM.

JUnit Test Results

Statistics Output

55 tests passed, 10 tests caused an error.

- com.commercehub.perspective.impl.housemodel.test.HousePerspectiveTest FAILED
 - testLoadPerspectivesStore passed (0.281 s)
 - testLoadPerspectivizedHouseWithPerspInsideRoom caused an ERROR (0.031 s)
 - testLoadPerspectivizedHouseWithPerspInsideCircuit passed (0.016 s)
 - testGetPropertyPaths caused an ERROR (0.0 s)
 - testGetPerspectiveIterator caused an ERROR (0.0 s)
 - testGetSubRealization caused an ERROR (0.0 s)
 - testGetSubRealizationIterator caused an ERROR (0.0 s)
 - testEditPerspective caused an ERROR (0.0 s)
 - testRemovePerspective caused an ERROR (0.0 s)
 - testGetContextualRelationship caused an ERROR (0.0 s)
 - testGetCircuitWithPerspective caused an ERROR (0.0 s)
 - testGetPerspective caused an ERROR (0.0 s)
- com.commercehub.perspective.impl.pojo.test.MetadataImplTest passed

JUnit Test Results Output

Samme eksempel uten bruk av inline interface-implementasjon

```
class StudentPrinterTester {
```

```
    public void print_KaltMed3Studenter_SkriverUtStudenterIRiktigRekkefølge() throws Exception {
```

```
        // arrange
```

```
        DummyPrinter printer = new DummyPrinter();  
        StudentPrinter eut = new StudentPrinter(printer);
```

```
        List<Student> studenter = new ArrayList<Student>();  
        studenter.add(new Student("Dag", "Langmyhr", "dag"));  
        studenter.add(new Student("Stein", "Gjessing", "steing"));  
        studenter.add(new Student("Eyvind", "Axelsen", "eyvinda"));
```

```
        // act
```

```
        eut.print(studenter);
```

```
        // assert
```

```
        if(!printer.utskrift.get(0).equals("Dag Langmyhr (dag)")) throw new RuntimeException("Feil");  
        if(!printer.utskrift.get(1).equals("Eyvind Axelsen (eyvinda)")) throw new RuntimeException("Feil");  
        if(!printer.utskrift.get(2).equals("Stein Gjessing (steing)")) throw new RuntimeException("Feil");
```

```
    }
```

```
}
```

```
class DummyPrinter implements IFysiskPrinter {  
    public List<String> utskrift = new ArrayList<String>();  
  
    public void print() {  
        utskrift.add(data);  
    }  
}
```

Oppsummert

- Å programmere handler om å forstå
- Kode leses mer enn den skrives – god lesbarhet er ekstremt viktig
- Bruk av SOLID-prinsippene kan fremme god lesbarhet og design
 - Single responsibility og Dependency inversion har vi sett på i dag
- Testbarhet oppnås ved
 - å ha klart definerte, enkle, ansvarsområder for klasser
 - å gjøre avhengighetene eksplisitte og utbyttbare
- Ved enhetstesting kan man bruke arrange, act, assert-mønsteret
 - Om man ønsker finnes det mange verktøy for å automatisere testkjøringen
 - Hjemmelekse: vil testen vi skrev gå bra, eller feiler den?





```
...y=d,this},a(window).on( 1000...  
a){"use strict";function b(b){return this.each(function(i) {  
c=function(b){this.element=a(b)};c.VERSION="3.3.7",c.TRANSITION_DURATIO...
```

```
...:b[0]}),g=a.Event("show.bs.tab",{relate...  
closest("li"),c),this.activate(h,h.pare...  
...]))))}}},c.prototype.activate=function...  
...d('[data-toggle="tab"]').attr("aria-exp...  
...ass("in")):b.removeClass("fade"),b.pare...  
...anded",!0),e&&e()}var g=d.find("> .act...  
...?g.one("bsTransitionEnd",f).emulateTra...  
...tor=c,a.fn.tab.noConflict=function(){re...  
...pi','[data-toggle="tab"]',e).on("click...  
...unction(){var d=a(this),e=d.data("bs.af...  
...options=a.extend({},c.DEFAULTS,d),thi...  
...ck.bs.affix.data-api",a.proxy(this.che...  
...ion());c.VERSION="3.3.7",c.RESET="affi...  
...scrollTop(),f=this.$element.offset(),...  
...his.unpin<=f.top)&&"bottom":!(e+g<=a-o...  
...ss("affix");var a=this.$target.scrollTop(),c.prototype.getPinnedOffset=functionio...  
...=function(){setTimeout(a.proxy(this.checkPosition,this),100...  
...d=this.options.offset,e=d.top,f=d.botto...  
...top(this.$element)) "c...  
...""
```

```
...ss("affix");var a=this.$target.scrollTop(),c.prototype.getPinnedOffset=functionio...  
...=function(){setTimeout(a.proxy(this.checkPosition,this),100...  
...d=this.options.offset,e=d.top,f=d.botto...  
...top(this.$element)) "c...  
...""
```