Feilmeldinger, kontrollflyt og void-metoder

Skjønne hvordan et program presist utføres
og forberede seg på håndtering av krøll

INF1000, uke2
Geir Kjetil Sandve

Krøll

- Programmeringskrøll:
  - *Programmet vil ikke kjøre (riktig),*
    og dette viser seg å skyldes detaljer som ikke har å gjøre med noe viktig (underliggende idé)
    - (feil som skyldes at datamaskinen er dum)
- Har du allerede rukket å oppleve mye av dette?
  - Det er riktig at den virkelige utfordringen i programmering er kreativ problemløsning
  - Men før man kommer til dette, må man få inn den nødvendige presisjonen

Vårt syn på krøll i kurset

- Essensen i faget er underliggende prinsipp
  - .. teknikaliteter er uviktig (for nå)
- Vi har forsøkt å fjerne kilder til krøll der vi kan
  - Men: noe krøll oppstår om man vil eller ikke - det er upresishetens natur
- Angrep er det beste forsvar
  - Vi fokuserer denne uka på krøll og praktikaliteter, med mål om at dere i størst mulig grad skal slippe å fokusere på det i de kommende uker..

Strategier for å håndtere krøll

- Kjenne til ulike feilmeldinger
  - Hvorfor de oppstår
  - Hvordan de utbedres
- Ha en presis forståelse av hvordan et program utføres (eksekveres)
  - Hvordan uttrykk (høyresiden på en linje) evalueres
  - Hvordan et program flyter fra linje til linje

Praktikaliteter

- Vi vil også presentere noen snarveier (forkortinger) når man skal kode
- Dette er egentlig uviktige teknikaliteter, men samtidig raskt å lære
- Formålet er igjen å bruke mindre energi på selve kodingen, og ha mer fokus tilgjengelig for problemløsning

Feil og feilmeldinger

Hva er en programfeil?

- Kompileringsfeil
  - Kompilatoren finner feil/svakhet (**javac** protesterer)
- Kjøretidsfeil
  - Ugyldig situasjon oppstår under kjøring (**java** protesterer)
- Logisk feil
  - Alt kjører, men resultatet er ikke i tråd med intensjonen (om man merker det eller ikke)

Skill mellom opprinnelig feil og følgefeil

- Ved kompileringsfeil
  - Første feilmelding er typisk den opprinnelige
  - I tillegg får man ofte mange etterfølgende feil, fordi kompilatoren kommer av sporet
- Ved kjøretidsfeil
  - En ugyldig tilstand kan skyldes en feil tidligere i koden enn der den merkes
- Ved logiske feil
  - Man kan få uønsket oppførsel ett sted, fordi man ikke tok høyde for bestemte utfall et annet sted

Bli kjent med programmeringsfeil

- Bruk tid på å fremprovosere flest mulig feil, og legg de bak øret!
- Senker frustrasjonsnivået betraktelig når feil oppstår
- Feilretting går mye raskere når man gjenkjenner feilmelding og hva den kan skyldes
- Det er mye å lære av å se hva som ikke fungerer

Vanlige kompileringsfeil

- {u2Kompileringsfeil1.java-u2Kompileringsfeil9.java}: eksempler på ulike kompileringsfeil

Vanlige kjøretidsfeil

- {u2Kjoretidsfeil1.java .. u2Kjoretidsfeil2.java}: eksempler på ulike kjøretidsfeil

Logiske feil

- {u2LogiskFeil1.java .. u2LogiskFeil2.java}: eksempler på logiske feil

Ulike praktikaliteter (av begrenset betydning)

Forkortinger rundt variabler

- Flere variabler kan deklareres på samme linje:
  - `int tall1, tall2, alder;`
- Man kan deklarere og gi verdi på samme linje:
  - `int alder = 6;`
- Forenklet skrivemåte for å oppdatere verdi:
  - `alder += 5`
  - (samme som: `alder = alder+5`)

Vær oppmerksom på

- Multiplikasjon må alltid angis eksplisitt med *
  - Feil: `int prod = 10 a;`
  - Riktig: `int prod = 10 * a;`
- = er noe helt annet enn ==
  - = brukes for å sette verdien til en variabel
  - == brukes for å sammenligne to verdier

Tegnsett

- En datamaskin representerer hver bokstav med en bestemt tallkode
- Det finnes ulike standarder for hvilke tall som representerer hvilke bokstaver
  - En klassisk standard er ASCII som støtter 128 tegn
  - En moderne standard er Unicode (UTF-8) som støtter over 100 000 tegn, og brukes på IFIs maskiner
  - Tegnsett er mer intrikat enn man kunne tro, og kan være kilde til mye krøll

Tegnsett

- Bruk av æ,ø,å i Java kan føre til krøll:
  - Bruk av æ,ø,å i variabelnavn o.l. i Java fører fort til krøll
  - Bruk av æ,ø,å i kommentarer eller tekststrenger burde gå bra, men kan også føre til problemer

Tegnsett

- Vi skal lære programmering,
ikke skrive stil,
og taper ingenting på å være uten æ,ø,å:
  - Skriv "forste aaret laerte jeg mye" i kodefilen,
og fokuser på det viktige i faget!

God programmerings-skikk

- Sørg for at programmet er oversiktlig å lese
  - Legg inn blanke linjer der det øker lesbarheten
  - Legg inn kommentarer der noe bør forklares (//)

God programmerings-skikk (forts.)

- Velg beskrivende navn fremfor korte navn:
  - **`antallStudenter`** er oftest bedre enn **`as`**
  - og i hvertfall ikke **`minVariabel`** for samme formål

God programmerings-skikk (forts.)

- Bruk store bokstaver for å skille de ulike delene av variabelnavnet (camelCase)
  - ikke: **`antallstudenter`**
  - men: **`antallStudenter`**

Kontrollflyt

Hvordan et program presist utføres

- Hvordan (i hvilken rekkefølge) utføres operasjoner helt detaljert på én enkelt linje
- Hvordan flyter (helt presist) et program fra linje til linje

Hva skjer innad på en linje?

- alder = 6;
  - veldig rett frem..
- alder = alder + 3;

Hva skjer innad på en linje?

- alder = 6;
  - veldig rett frem..
- alder = alder + 3;
  - Gjør ferdig høyresida for likhetstegnet først

Hva skjer innad på en linje?

- alder = 6;
  - veldig rett frem..
- alder = alder + 3;

6

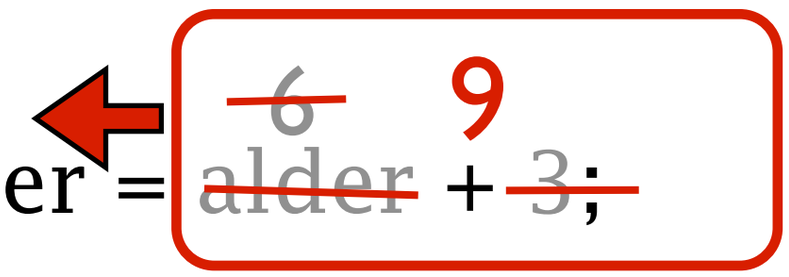
  - Gjør ferdig høyresida for likhetstegnet først
  - Alle verdier er på høyresida slik de var før denne linja (alder er altså 6)

Hva skjer innad på en linje?

- alder = 6;
  - veldig rett frem..
- alder = ~~6~~ 9 ~~alder~~ + ~~3~~;
  - Gjør ferdig høyresida for likhetstegnet først
  - Alle verdier er på høyresida slik de var før denne linja (alder er altså 6)
  - Regner ut 6+3 og får 9

Hva skjer innad på en linje?

- alder = 6;
  - veldig rett frem..
- alder = alder + 3;



  - Gjør ferdig høyresida for likhetstegnet først
  - Alle verdier er på høyresida slik de var før denne linja (alder er altså 6)
  - Regner ut 6+3 og får 9
  - Setter til slutt verdien 9 inn i alder

Videre om evaluering av uttrykk

- Visse operasjoner gjøres alltid før andre
  - Java gjør f.eks.uansett ganging før plussing
- Følgende gir altså samme resultat 33:
  - `alder = 5 * 3 + 18`

15

33

Videre om evaluering av uttrykk

- Visse operasjoner gjøres alltid før andre
  - Java gjør f.eks.uansett ganging før plussing
- Følgende gir altså samme resultat 33:
  - `alder = 5 * 3 + 18` 15 33
  - `alder = 18 + 5 * 3`

Videre om evaluering av uttrykk

- Visse operasjoner gjøres alltid før andre
  - Java gjør f.eks.uansett ganging før plussing
- Følgende gir altså samme resultat 33:
  - `alder = 5 * 3 + 18` (15, 33)
  - `alder = 18 + 5 * 3`
- Blir tydeligere med paranteser, bruk det!
  - alder = (5 * 3) + 18

Videre om evaluering av uttrykk (forts.)

- For noen formål må man uansett ha paranteser
  - `alder = (3+2) * 3 + 18`

5 15 33

Videre om evaluering av uttrykk (forts.)

- For noen formål må man uansett ha paranteser
  - `alder = (3+2) * 3 + 18`

5 15 33

- Og for å gjøre det samme tydeligere - nøsting av paranteser er definitivt lovlig:
  - alder = ((3+2) * 3) + 18

Videre om evaluering av uttrykk (forts.)

- For noen formål må man uansett ha paranteser
  - alder = ~~(3+2)~~ ~~* 3~~ ~~+ 18~~ (5) (15) (33)
- Og for å gjøre det samme tydeligere - nøsting av paranteser er definitivt lovlig:
  - alder = ((3+2) * 3) + 18
- I praksis er det ikke tall man putter inn på slik måte, men variabler:
  - alder = ((bachelor+master) * antallFagfelt) + barndom

Boolske uttrykk
(Sammensatte sannheter)

- Vi så et eksempel forrige uke:
  - `if (alder>=3 && alder<6) {println("liten")}`
  - && betyr "og" - altså må begge deler være sant
- Man kan også sjekke enten eller med ||
  - if (alder<18 || alder>66) {println("Halv pris")}

Boolske uttrykk (forts)

- Utrykk kan naturligvis inneholde ulike variabler
  - if (alder>80 || dagerTilTermin<10) {println("Ta mitt sete!")}
- Og så kan utrykkene være nøstede
  - if **((**dagerTilTermin>180 && tidspunkt=="morgen"**) ||** **(**alder<6 && antallKilometerBiltur>30**))** {println("du er sikkert kvalm!")}

Hvordan et program flyter fra linje til linje

- Dette er temmelig enkelt for det vi har lært frem til nå (endrer seg om en halvtime..)
- Hovedregel:
  - Gjør ferdig en linje, deretter gå til linjen nedenfor
- Ved en if-setning
  - Dersom det som testes er sant:
    gå til første linje i blokken
  - ellers:
    hopp over blokken

Hvordan et program flyter fra linje til linje

- Ved en if-else-setning
- Dersom det som testes er sant:
  gå til første linje i blokken etter if
- ellers:
  gå til første linje i blokken etter else

Se hvordan et program kjører vha en **debugger**

- Med et program som heter "debugger" kan man kjøre programmet linje for linje
- Et kommandolinjeverktøy "jdb" for dette følger med Java, men er upraktisk
- Det er greit å gjøre i f.eks. DrJava:
  Skru på debugger, sett et "breakpoint", kjør

Kjøre linje for linje i debugger

- {u2Omkrets.java}

Etterlign kjøring, med blyant og papir

- Gjør manuelt det samme som debuggeren ville gjort, linje for linje
  - Kan gjøres i hodet, men enklere på papir (print ut koden og bruk blyant)
  - Vær presis - når man debugger er hver detalj viktig

Følge et program, linje for linje

```
public class u20mkrets{
  public static void main(String[] args) {
    int lengde=7;
    int bredde=4;
    int omkrets;

    if (lengde==bredde){
      omkrets = 4*lengde;
    } else {
      omkrets = (2*lengde) + (2*bredde);
    }
    System.out.println("Omkrets: " + omkrets) ;
  }
}
```

Følge et program, linje for linje

```
public class u20mkrets{
  public static void main(String[] args) {
    int lengde=7;
    int bredde=4;
    int omkrets;

    if (  7    ==bredde){
      omkrets = 4*lengde;
    } else {
      omkrets = (2*lengde) + (2*bredde);
    }
    System.out.println("Omkrets: " + omkrets) ;
  }
}
```

Følge et program, linje for linje

```
public class u20mkrets{
  public static void main(String[] args) {
    int lengde=7;
    int bredde=4;
    int omkrets;

    if (  7   ==   4  ){
      omkrets = 4*lengde;
    } else {
      omkrets = (2*lengde) + (2*bredde);
    }
    System.out.println("Omkrets: " + omkrets) ;
  }
}
```

Følge et program, linje for linje

```
public class u20mkrets{
  public static void main(String[] args) {
    int lengde=7;
    int bredde=4;
    int omkrets;

    if (  7   ==   4  ){
      omkrets = 4*lengde;
    } else {
      omkrets = (2*  7   ) + (2*bredde);
    }
    System.out.println("Omkrets: " + omkrets) ;
  }
}
```

Følge et program, linje for linje

```
public class u20mkrets{
  public static void main(String[] args) {
    int lengde=7;
    int bredde=4;
    int omkrets;

    if (  7   ==   4  ){
      omkrets = 4*lengde;
    } else {
      omkrets = (   14   ) + (2*bredde);
    }
    System.out.println("Omkrets: " + omkrets) ;
  }
}
```

Følge et program, linje for linje

```
public class u20mkrets{
  public static void main(String[] args) {
    int lengde=7;
    int bredde=4;
    int omkrets;

    if (  7   ==   4  ){
      omkrets = 4*lengde;
    } else {
      omkrets = (   14   ) + (2*   4  );
    }
    System.out.println("Omkrets: " + omkrets) ;
  }
}
```

Følge et program, linje for linje

```
public class u20mkrets{
  public static void main(String[] args) {
    int lengde=7;
    int bredde=4;
    int omkrets;

    if (  7   ==   4  ){
      omkrets = 4*lengde;
    } else {
      omkrets = (   14   ) + (    8   );
    }
    System.out.println("Omkrets: " + omkrets) ;
  }
}
```

Følge et program, linje for linje

```
public class u20mkrets{
  public static void main(String[] args) {
    int lengde=7;
    int bredde=4;
    int omkrets;

    if (  7   ==   4  ){
      omkrets = 4*lengde;
    } else {
      omkrets =          22              ;
    }
    System.out.println("Omkrets: " + omkrets) ;
  }
}
```

Følge et program, linje for linje

```
public class u20mkrets{
  public static void main(String[] args) {
    int lengde=7;
    int bredde=4;
    int omkrets;

    if (  7   ==   4  ){
      omkrets = 4*lengde;
    } else {
      omkrets =          22              ;
    }
    System.out.println("Omkrets: " +   22   ) ;
  }
}
```

Metoder

Vi trenger mer struktur!

- Vi har frem til nå skrevet programmer linje for linje nedover under "static void main"
- Realistiske program er imidlertid ofte tusener eller millioner av linjer!
  - Ingen kan ha oversikt over en flat liste på mange tusen (eller millioner) av linjer
- Et første nivå av strukturering er **metode**: en **navngitt blokk** med kodelinjer, som kan **kalles** og **tilpasses**

Ulike versjoner av metoder

- Metoder kommer i ulike versjoner, av gradvis økende kompleksitet
- Vi vil introdusere de involverte aspektene stegvis
  - Først i dag: Statisk void-metode **uten** parametre
  - Også i dag: Statisk void-metode **med** parametre
  - Om to uker: Statisk metode med **returverdi**
  - Litt senere i høst: **Instans**-metode (OO)

Hvordan kan en metode se ut

```
public static void mittMetodeNavn(){
   kodelinje1;
   kodelinje2;
   ...
}
```

For å kjøre alle kodelinjene i metoden (kalle den):

```
mittMetodeNavn();
```

Eksempel på metode: kodeblokk

```
{
  System.out.println("Jeg sier dette bare en gang!");
  System.out.println("Da var jeg ferdig!");
}
```

Eksempel på metode: navn

```
static void giBeskjed(){
  System.out.println("Jeg sier dette bare en gang!");
  System.out.println("Da var jeg ferdig!");
}
```

Eksempel på metode: kall

```
static void giBeskjed(){
  System.out.println("Jeg sier dette bare en gang!");
  System.out.println("Da var jeg ferdig!");
}


  giBeskjed();
```

Eksempel på metode: ramme rundt kallet

```
static void giBeskjed(){
  System.out.println("Jeg sier dette bare en gang!");
  System.out.println("Da var jeg ferdig!");
}

public static void main(String[] args) {
  giBeskjed();
}
```

Eksempel på metode: ramme rundt hele koden

```
public class u2VoidMetodeUtenParameter1{
   static void giBeskjed(){
      System.out.println("Jeg sier dette bare en gang!");
      System.out.println("Da var jeg ferdig!");
   }

   public static void main(String[] args) {
      giBeskjed();
   }
}
```

Kontrollflyt og metoder

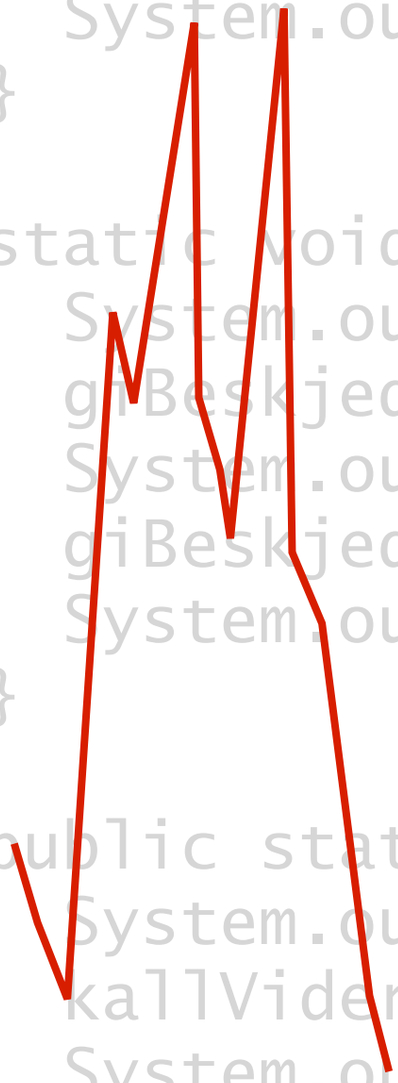
```
public class u2VoidMetodeUtenParameter{
  static void giBeskjed(){
    System.out.println("Jeg sier dette bare en gang!");
  }

  static void kallVidere(){
    System.out.println("Her kommer det");
    giBeskjed();
    System.out.println("Og en gang til");
    giBeskjed();
    System.out.println("Det var ikke mye");
  }

  public static void main(String[] args) {
    System.out.println("Hei, jeg vil si deg noe");
    kallVidere();
    System.out.println("Fikk du det likevel med deg?");
  }
}
```

Kontrollflyt og metoder

```
public class u2VoidMetodeUtenParameter{
  static void giBeskjed(){
    System.out.println("Jeg sier dette bare en gang!");
  }

  static void kallVidere(){
    System.out.println("Her kommer det");
    giBeskjed();
    System.out.println("Og en gang til");
    giBeskjed();
    System.out.println("Det var ikke mye");
  }

  public static void main(String[] args) {
    System.out.println("Hei, jeg vil si deg noe");
    kallVidere();
    System.out.println("Fikk du det likevel med deg?");
  }
}
```

Kontrollflyt og metoder

```
public class u2VoidMetodeUtenParameter{
  static void giBeskjed(){
    System.out.println("Jeg sier dette bare en gang!");
  }

  static void kallVidere(){
    System.out.println("Her kommer det");
    giBeskjed();
    System.out.println("Og en gang til");
    giBeskjed();
    System.out.println("Det var ikke mye");
  }

  public static void main(String[] args) {
    System.out.println("Hei, jeg vil si deg noe");
    kallVidere();
    System.out.println("Fikk du det likevel med deg?");
  }
}
```



Statisk void-metode **med** parametre

- Void-metoden vi så på tidligere gjorde alltid eksakt det samme når den ble kallet
  - Det er sjelden av nytte!
- For å være nytig må en slik metode kunne **tilpasses**
  - Det gjør vi ved å sende inn **parametre**

Din første metode med parametre

- **`println`** er en metode hvor utfallet tilpasses!
- `System.out.println(String text)`: skriver verdien i `text` til skjermen
  - Variabelen text er en **parameter**
  - Verdien vi gir inn (`hallo verden`) når vi kaller `println` er et **argument**
- Parameter og argument er to sider av det samme
  - Parameter: **variabel** i metode som tar i mot verdi
  - Argument: **verdi** sendt inn når metode kalles

Metoder håndterer også redundans

- Man har ofte behov for (omtrent) samme funksjonalitet ulike steder i en kode
- Man ønsker da ikke å duplisere koden
  - Minsker oversiktlighet av kode
  - Endringer og rettinger må utføres mange steder
- Man samler i stedet funksjonaliteten i en metode og kaller metoden der den trengs
  - Dersom det er noe variasjon i hva man trenger, representerer man det som varierer med en parameter

Metode med parametre

- {u2VoidMetodeParameter1.java-u2VoidMetodeParameter4.java}

Emacskurs

Fagutvalget ved Institutt for informatikk arrangerer kurs i Emacs for folk som aldri har brukt Emacs før.

Hvis du er ny Emacs-bruker, eller fortsatt skriver Java og C i GEdit så er dette kurset for deg.

Kurset blir holdt av Lars Tveito på Simula,
3. september fra kl 16.15 til 18.00

Det blir pizza og kos i Escape i etterkant av kurset

Fagutvalget ved Institutt for informatikk

Oppsummering

- Feil kan fanges opp ved kompilering, ved kjøring, eller bare gi uønsket resultat
  - Å kjenne til vanlige feilmeldinger senker frustrasjon når feil uunngåelig oppstår
- Programkode kjøres på en presist definert måte innad i en linje, og fra linje til linje
  - Å leke debugger med blyant og papir er nyttig!
- Metoder strukturerer kode og unngår redundans
  - Kan tilpasses med parametre