Mer objektorientert programmering

- Klasser og objekter
- En klasse med konstruktør og en array
- Innkapsling
- En klasse med datafil og kommandoløkke
- Klassen ArrayList

Klasser og objekter

Det er viktig å ha klart for seg hva som er hva:

En klasse er en *arbeidsbeskrivelse* for hvordan man skal lage objekter. Det finnes alltid nøyaktig ett eksemplar av klassen når programmet kjører.

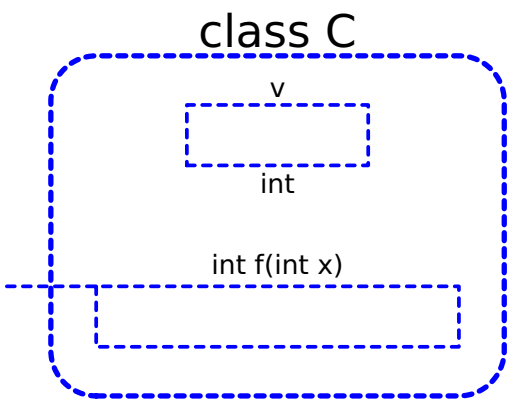
Objekter er *instanser* laget utifra beskrivelsen i en klasse. Når programmet starter, er det ingen objekter; de lages etter hvert med **new**. Derfor kan det være vilkårlig mange objekter av hver klasse.

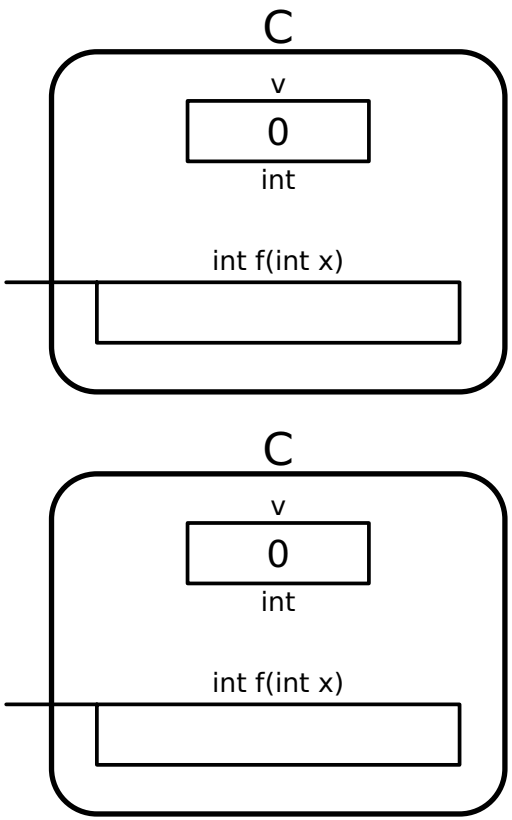
Objektvariabler og -metoder

```
class C {
  private int v;

  int f(int x) {
    :
  }
}

... new C() ... new C() ...
```



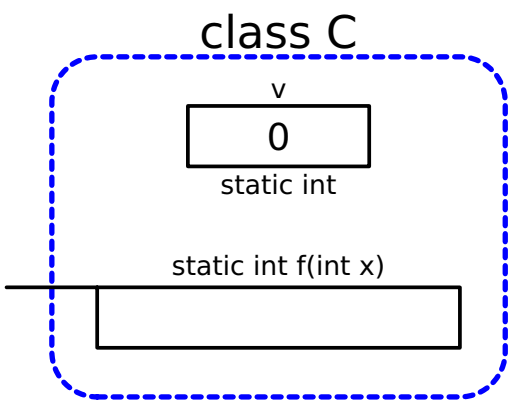


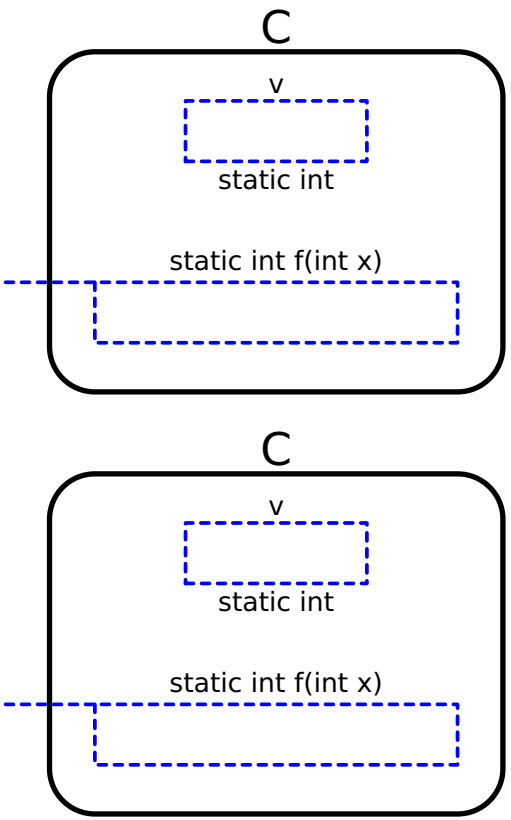
Klassevariabler og -metoder

```
class C {
  private static int v;

  static int f(int x) {
    :
  }
}

... new C() ... new C() ...
```





Eksempel 1: En timer

En **timer** varsler når en gitt tid har gått.

Steg 1: Hvilke klasser trenger vi?

Svaret er innlysende: class Timer.

Steg 2: Hva er grensesnittet?

Hva hvis vi jobber med flere ting samtidig? Hadde det ikke vært fint å ha flere klokker i den samme timer? Tenk om vi bare kunne skrive

```
new Timer(4)
```

og så hadde vi fire klokker.

Dette gjøres i Java med en **konstruktør** med parametre:

- Konstruktør med antall klokker:
  Timer(int antallKlokker) {...}

(En konstruktør ligner på en metode, men har intet navn, kun klassenavnet.)

Resten av grensesnittet

- Velge klokke:
  void nesteKlokke() {...}
- Stille tiden for valgte klokke:
  void stillTid(int m, int s) {...}
- Starte valgte klokke:
  void start() {...}
- Stoppe valgte klokke:
  void stopp() {...}
- Ta imot klokkesignal hvert sekund:
  void tikk() {...}
- Vise gjenværende tid for valgte klokke:
  public String toString() {...}

Å kunne skrive ut seg selv

Det er veldig nyttig å kunne skrive ut seg selv; alle klasser bør kunne det. Dette ordnes i Java ved å definere

public String toString() {...}

Når vi skjøter sammen tekster med +, er det faktisk elementenes toString()-resultater vi bruker.

Eksempel

```
Timer t = new Timer(1);
System.out.println("Tiden er " + t.toString());
System.out.println("Tiden er " + t);  // Det samme!
```

Steg 3: Representasjonen

Hva må vi lagre?

- Tiden i minutter og sekunder for hver klokke:
  private int[] min;
  private int[] sek;
- Hvilken klokke ser vi nå på skjermen?
  private int denneKlokke;
- Hvilke klokker er aktive nå?
  private boolean[] aktiv;

Steg 4: Skriv ferdig grensesnittmetodene

```
class Timer {
  private int[] min;
  private int[] sek;
  private boolean[] aktiv;
  private int denneKlokke;

  Timer(int antallKlokker) {
    min = new int[antallKlokker];
    sek = new int[antallKlokker];
    aktiv = new boolean[antallKlokker];
    denneKlokke = 0;
  }

  public String toString() {
    return "[" + denneKlokke + "] " +
      min[denneKlokke] + ":" +
      sek[denneKlokke];
  }
```

Steg 4

```
void stillTid(int m, int s) {
  min[denneKlokke] = m;
  sek[denneKlokke] = s;
}

void start() {
  aktiv[denneKlokke] = true;
}

void stopp() {
  aktiv[denneKlokke] = false;
}

void nesteKlokke() {
  denneKlokke++;
  if (denneKlokke >= aktiv.length) {
    denneKlokke = 0;
  }
}
```

Steg 4

```
    void tikk() {
      for (int i = 0;  i < aktiv.length;  i++) {
        if (aktiv[i]) {
          if (sek[i] > 0) {
            sek[i]--;
          } else if (min[i] > 0) {
            min[i]--;  sek[i] = 59;
          }

          if (min[i]==0 && sek[i]==0) {
            System.out.print("ALARM #" + i);
            aktiv[i] = false;
          }
        }
      }
    }
}
```

Et testprogram

```
class TestTimer {
    public static void main(String[] arg) {
        Timer t = new Timer(2);

        t.stillTid(0, 10);  t.nesteKlokke();
        t.stillTid(0, 20);

        System.out.println(t);  t.nesteKlokke();  System.out.println(t);

        t.start();  t.nesteKlokke();
        t.start();

        for (int i = 1;   i <= 25;  i++) {
            System.out.print(".");  t.tikk();
        }
        System.out.println();
    }
}
```

Kjøring

```
$ java TestTimer
[1] 0:20
[0] 0:10
..........ALARM #0..........ALARM #1.....
```

Innkapsling

I klassen Timer valgte vi å representere tiden med to verdier: minutter og sekunder. Kunne vi valgt noe annet?

I klasser er svaret: selvfølgelig!

Hva må endres for å lagre tiden bare som sekunder?

Innkapsling er en flott egenskap ved objektorientert programmering

```
class Timer {
  private int[] min;
  private int[] sek;
  private boolean[] aktiv;
  private int denneKlokke;

  Timer(int antallKlokker) {
    min = new int[antallKlokker];
    sek = new int[antallKlokker];
    aktiv = new boolean[antallKlokker];
    denneKlokke = 0;
  }

  public String toString() {
    return "[" + denneKlokke + "] " +
      min[denneKlokke] + ":" +
      sek[denneKlokke];
  }

  void stillTid(int m, int s) {
    min[denneKlokke] = m;
    sek[denneKlokke] = s;
  }

  void start() {
    aktiv[denneKlokke] = true;
  }

  void stopp() {
    aktiv[denneKlokke] = false;
  }
```

```
class Timer2 {
  private int[] sek;
  private boolean[] aktiv;
  private int denneKlokke;

  Timer2(int antallKlokker) {
    sek = new int[antallKlokker];
    aktiv = new boolean[antallKlokker];
    denneKlokke = 0;
  }

  public String toString() {
    return "[" + denneKlokke + "] " +
      (sek[denneKlokke]/60) + ":" +
      (sek[denneKlokke]%60);
  }

  void stillTid(int m, int s) {
    sek[denneKlokke] = 60*m + s;
  }

  void start() {
    aktiv[denneKlokke] = true;
  }

  void stopp() {
    aktiv[denneKlokke] = false;
  }
```

Innkapsling er en flott egenskap ved objektorientert programmering

```
  void nesteKlokke() {
    denneKlokke++;
    if (denneKlokke >= aktiv.length) {
      denneKlokke = 0;
    }
  }

  void tikk() {
    for (int i = 0;  i < aktiv.length;  i++) {
      if (aktiv[i]) {
        if (sek[i] > 0) {
          sek[i]--;
        } else if (min[i] > 0) {
          min[i]--;  sek[i] = 59;
        }

        if (min[i]==0 && sek[i]==0) {
          System.out.print("ALARM #" + i);
          aktiv[i] = false;
        }
      }
    }
  }
}
```

```
  void nesteKlokke() {
    denneKlokke++;
    if (denneKlokke >= aktiv.length) {
      denneKlokke = 0;
    }
  }

  void tikk() {
    for (int i = 0;  i < aktiv.length;  i++) {
      if (aktiv[i]) {
          if (sek[i] > 0) {
              sek[i]--;
          }

        if (sek[i] == 0) {
          System.out.print("ALARM #" + i);
          aktiv[i] = false;
        }
      }
    }
  }
}
```

Dette er det fine med **innkapsling**:

Innkapsling

Siden representasjonen og innmaten i grensesnittmetodene er innkapslet, kan de endres uten at det affiserer kode som benytter klassen.

Eksempel 2: Et DVD-arkiv

Vi ønsker oss et arkiv over DVDene våre, med muligheter til å redigere arkivet.

Designvalg

- Vi vil lagre arkivet på disk mellom hver gang vi benytter det.
- Vi vil ha et program der vi kan gi kommandoer som så blir utført av programmet.

Lesing fra fil

Linux-programmet cat skriver ut innholdet av en tekstfil; dette programmet gjør det samme:

```
import java.util.Scanner;
import java.io.*;

class Cat {
    public static void main(String[] arg) throws Exception {
        File f = new File(arg[0]);
        if (f.exists()) {
            Scanner s = new Scanner(f);
            while (s.hasNextLine()) {
                String t = s.nextLine();
                System.out.println(t);
            }
            s.close();
        }
    }
}
```

Skriving til fil

```
import java.io.*;

void m() throws Exception {
  PrintWriter f = new PrintWriter("filnavn");

  ... f.print("A"); ... f.println("Z"); ...
  f.close();
}
```

Fase 1: Hvilke klasser trenger vi?

Det virker naturlig å ha tre klasser:

- class DVD for å lagre data om en DVD
- class DVDArkiv for å representere et arkiv
- class TestDVDArkiv for å sjekke det vi har skrevet

Klassen DVD

Fase 2: Grensesnitt

- Konstruktør med angitt DVD-tittel:
  DVD(String n) {...}
- Vise tittel:
  public String toString() {...}

Fase 3: Representasjon

- Navnet på DVDen:
  private String navn;

Fase 4: Implementasjon

DVD.java

```
class DVD {
    private String navn;

    DVD(String n) {
        navn = n;
    }

    public String toString() {
        return navn;
    }
}
```

Klassen DVDArkiv

Steg 2: Grensesnitt

- Konstruktør med filnavnet der arkivet lagres:
  DVDArkiv(String navn) { ... }
- Navnet på arkivet (til f eks utskrift):
  public String toString() { ... }
- Les arkivet inn fra disk:
  void lesArkiv() throws Exception { ... }
- Skriv arkivet tilbake til disk:
  void skrivArkiv() throws Exception { ... }
- Utfør kommandoer brukeren gir:
  void utfoerKommandoer() { ... }

Steg 3: Representasjon

- Navnet på arkivet (dvs filnavnet der det ligger lagret):
  private String arkivnavn;
- Alle DVDene:
  private DVD[] arkiv;
- Antall DVDer:
  private int antall;

Kommandoløkke

Mange programmer som kommuniserer bruker, gjør det med en **kommandoløkke**: brukeren gir én og én kommando som programmet utfører.

```
while (true) {
  «Be om en kommando.»
  «Les en kommando.»
  if (kommando.startsWith("...")) {
    «Utfør kommandoen.»
  } else if (kommando.startsWith("...")) {
    «Utfør kommandoen.»
  } else {
    «Gi melding om ulovlig kommando.»
  }
}
```

Husk at minst én kommando må avslutte løkken med f eks en **return**.

Steg 4: Resten av koden

DVDArkiv.java

```
import java.util.Scanner;
import java.io.*;

class DVDArkiv {
  private String arkivnavn;
  private DVD[] arkiv = new DVD[1000];
  private int antall = 0;

  DVDArkiv(String navn) {
    arkivnavn = navn;
  }

  public String toString() {
    return "DVD-arkivet " + arkivnavn;
  }
```

class DVDArkiv

```
void lesArkiv() throws Exception {
  File f = new File(arkivnavn);

  if (f.exists()) {
    Scanner s = new Scanner(f);

    while (s.hasNextLine()) {
      arkiv[antall] = new DVD(s.nextLine());
      antall++;
    }
    s.close();
    System.out.println("Arkivet " + arkivnavn + " er lest.");
  } else {
    System.out.println("Nytt arkiv " + arkivnavn + " opprettet.");
  }
}
```

class DVDArkiv

```
void skrivArkiv() throws Exception {
  PrintWriter p = new PrintWriter(arkivnavn);

  for (int i = 0;  i < antall;  i++) {
    p.println(arkiv[i]);
  }
  p.close();
}
```

class DVDArkiv

```
void utfoerKommandoer() {
  Scanner s = new Scanner(System.in);

  while (true) {
    System.out.println();
    System.out.println("Gi en kommando:");
    System.out.println("  A (Avslutt)");
    System.out.println("  N (Ny DVD)");
    System.out.println("  V (Vis oversikt)");
    System.out.print("Kommando: ");

    String kommando = s.nextLine();
    if (kommando.startsWith("A")) {
      return;
    } else if (kommando.startsWith("N")) {
      System.out.print("DVDens navn: ");
      arkiv[antall] = new DVD(s.nextLine());
      antall++;
    } else if (kommando.startsWith("V")) {
      for (int i = 0;  i < antall;  i++) {
          System.out.println(i + ". " + arkiv[i]);
      }
    } else {
      System.out.println("'" + kommando +
        "' er en ulovlig kommando!");
    }
  }
```

Testprogrammet

TestDVDArkiv.java

```
class TestDVDArkiv {
    public static void main(String[] arg) throws Exception  {
        DVDArkiv a = new DVDArkiv("dvd-arkiv.text");

        a.lesArkiv();
        a.utfoerKommandoer();
        a.skrivArkiv();
    }
}
```

Kjøringen

```
$ javac TestDVDArkiv.java
$ java TestDVDArkiv
Arkivet dvd-arkiv.text er lest.

Gi en kommando:
  A (Avslutt)
  N (Ny DVD)
  V (Vis oversikt)
Kommando: Vis
0. Ringenes herre 1-3
1. Harry Potter 1-7

Gi en kommando:
  A (Avslutt)
  N (Ny DVD)
  V (Vis oversikt)
Kommando: Ny
DVDens navn: Hobbiten 1-3
Gi en kommando:
  A (Avslutt)
  N (Ny DVD)
  V (Vis oversikt)
Kommando: Vis
0. Ringenes herre 1-3
1. Harry Potter 1-7
2. Hobbiten 1-3

Gi en kommando:
  A (Avslutt)
  N (Ny DVD)
  V (Vis oversikt)
Kommando: Avslutt
```

Peker til ingenting

Når vi opprettet arrayen

```
private DVD[] arkiv = new DVD[1000];
```

hva peker alle elementene i arkiv på før vi tilordner noe?

Svaret er: ingenting. I Java heter det **null**.

Et eksempel med feil

```
class Feil {
    public static void main(String[] arg) {
        DVD[] arkiv = new DVD[10];

        System.out.println("Min eldste DVD er " +
            arkiv[0].toString());
    }
}
```

```
$ javac Feil.java
$ java Feil
Exception in thread "main" java.lang.NullPointerException
at Feil.main(Feil.java:5)
```

Da må vi finne ut hvor feilen skjedde og hvorfor.

Arrayer

Arrayer er en usedvanlig nyttig mekanisme så alle programmeringsspråk har dem.

- Enkel og klar notasjon: **a[i]**
- Rask i bruk
- Må oppgi antallet elementer når arrayen lages
  - Hvis antallet er for stort, sløser vi med plassen.
  - Hvis det er for lite, krasjer programmet

Klassen ArrayList

Bibliotekklassen ArrayList er et forsøk på forbedre arrayer:

```
String[] a = new String[100];
int antall = 0;

a[0] = "Bergen";  antall++;
a[1] = "Oslo";    antall++;

s = a[1];

n = antall;
```

```
import java.util.ArrayList;

ArrayList<String> a =
    new ArrayList<>();

a.add("Bergen");
a.add(1, "Oslo");

s = a.get(1);

n = a.size();
```

Kan noen forbedre arrayer?

- Vi trenger ikke oppgi størrelsen; et ArrayList-objekt vil øke i størrelse automatisk.
- Litt kronglete notasjon.
- Fungerer best for å lagre pekere; verditypene **int**, **float** etc må gis særbehandling.

Det er opp til deg som programmerer å velge hva du foretrekker.