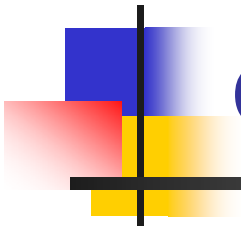
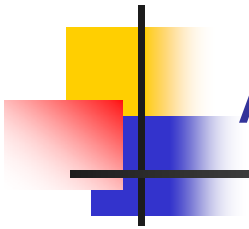


OOP: Et stort eksempel

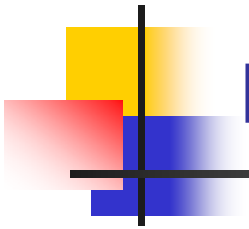


Administasjon av hopprenn

Konkret eksempel på OOP-program. Programmet skal

- registrere navn og idrettslag til skihoppere
- trekke startlisten til første omgang
- lese inn lengde og 5 stilkarakterer for hvert hopp
- beregne poengsum og skrive ut resultatlisten
- beregne startrekkefølgen i 2. omgang ved å snu resultatlisten fra 1. omgang
- kunne simulere hopprennet ut fra tilfeldige tall

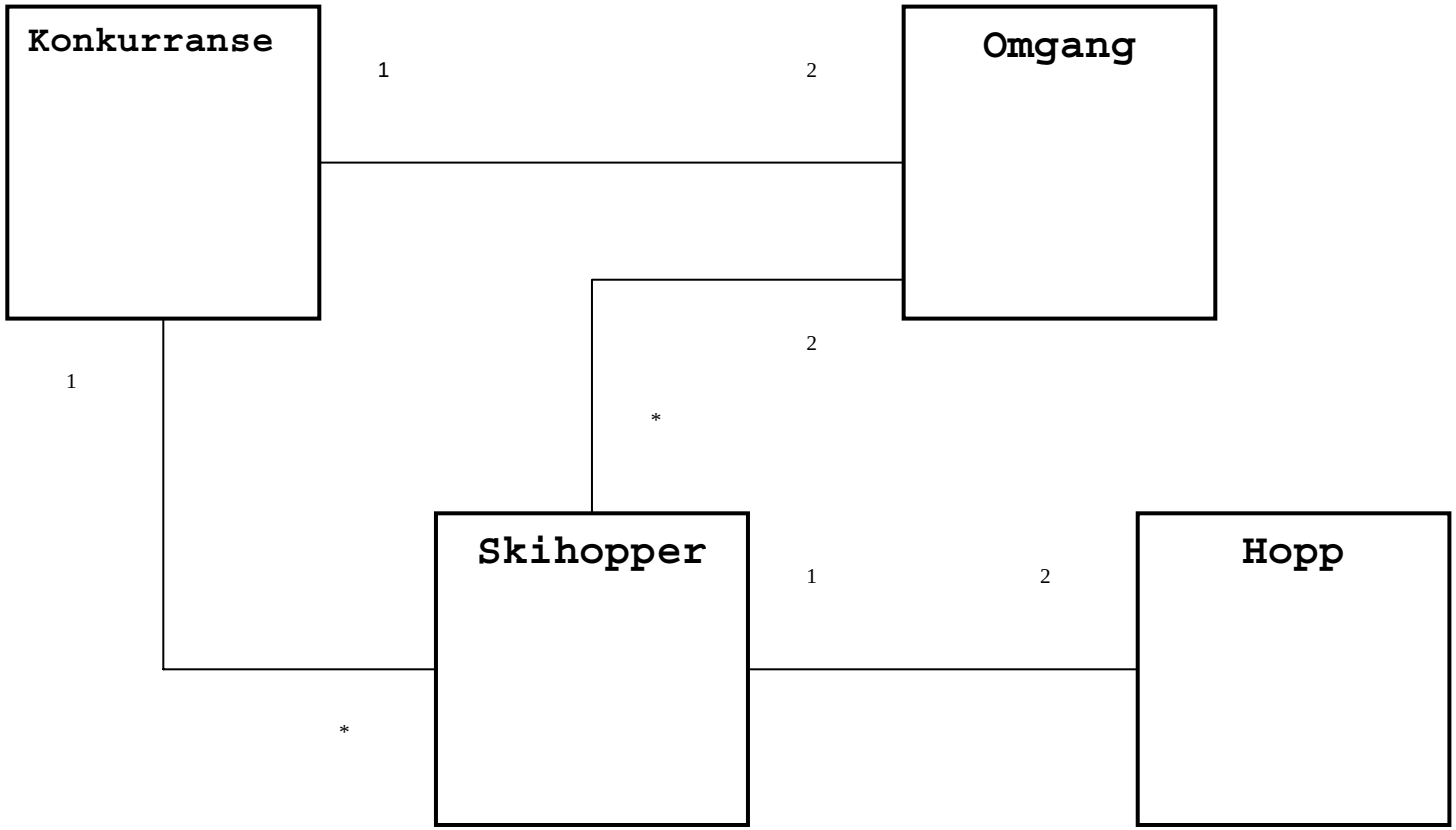
Metoden for beregning av poengsum ut fra lengde og stilkarakterer er beskrevet i oppgave 5.9 i læreboka.



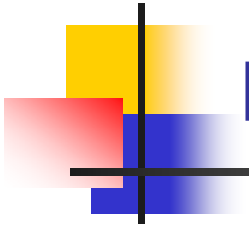
Hva er objektene?

- I et generelt tilfelle vil objektene i en OOP-modell motsvare både konkrete og abstrakte ting i verden
- I et hopprenn er det mange *skihoppere*, disse er de konkrete objektene.
- Vi har også en rekke *hopp* som, om de ikke er helt konkrete, i alle fall er noe vi kan referere entydig til.
- Litt mer abstrakt består hopprennet av to *omganger*.
- Vi trenger også å kunne henvise til selve *konkurransen*

Klassediagram av hopprenn-systemet



Hva består en klasse av?



Klassenavn
Variable
Konstrukør
Metoder

Datastruktur – der informasjonen lagres

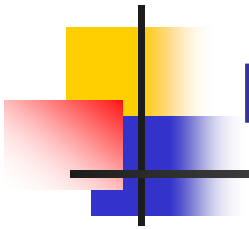
Initialisering av datastrukturen når objektet opprettes

Tjenestene som tilbys

class Hopprenn og main-metoden

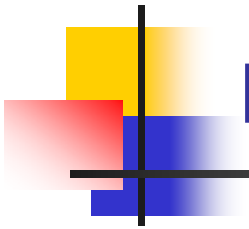
```
class Hopprenn {
    public static void main( String[] args ){
        Konkurranse rennadm = new Konkurranse();
        rennadm.kommandoløkke();
    }
}
```

- Det opprettes et objekt av klassen Konkurranse som heter rennadm. Dette skjer idet setningen new Konkurranse() utføres.
- Når objektet rennadm opprettes, utføres en bestemt metode i class Konkurranse som kalles *konstruktøren*.
- Vi kaller metoden til objektet rennadm. Merk at metoden har en adresse (nemlig rennadm) og at denne er ansvarlig for at kommandoene utføres.
- Metoden kommandoløkke() ligger i class Konkurranse.



Konstruktøren

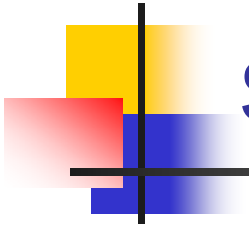
- Konstruktøren heter alltid det samme som klassen
- Den skrives uten typeangivelse. Konstruktøren i Konkurranse-klassen heter kun Konkurranse()
- Den utføres en og bare en gang for hvert objekt (nemlig når det opprettes)
- Kan utelates fra klassen
- Brukes normalt til initialiseringer



Konkurranse-klassen: Hovedmeny

I hovedmenyen skal vi foreta registrering av nye skihoppere og kunne starte omganger:

```
*** MENY ***
0. Avslutt
1. Registrer ny deltager
2. Trekning av startnummer
3. List alle deltagere
4. Første omgang
5. Andre omgang
6. Generer fiktive deltagere
```

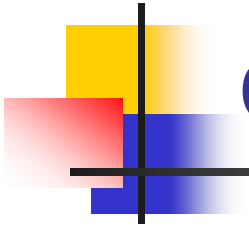


Skisse til klassen

```
import easyIO.*;
class Konkurranse {


    Konkurranse() {
        System.out.println("HOPP-PROGRAM VERSJON 1.0");
    }
    void kommandoløkke(){
        ...
    }
    // Her kommer alle de andre metodene i class Konkurranse
}
```

void kommandoløkke()

```
do {
    System.out.print("\nValg (9 for Meny): ");
    valg = tast.inInt();
    switch(valg){
        case 0: System.out.println("Programmet avslutter");
                System.out.println(); break;
        case 1: registrerDeltager(); break;
        case 2: trekning(); break;
        case 3: listDeltagere(); break;
        case 4: /* kode følger siden */ break;
        case 5: /* kode følger siden */ break;
        case 6: autogenerer(); break;
        case 9: skrivMeny(); break;
        default: System.out.println("Du tastet feil");
    }
} while (!(valg == 0));
```



Grunnleggende datastruktur

Vi må ha en datastruktur som effektivt tillater å

- legge inn data om skihoppere
- assosiere skihoppere med hopp
- assosiere startlister og resultatlister med skihoppere (men ikke nødvendigvis motsatt)

➔ Vi trenger å ha lett tilgang til informasjonen om hopperne

➔ Vi trenger å gruppere hoppere i hver omgang

Den enkleste løsningen er å gruppere skihoppere i arrayer:

- arrayene administreres innen hver omgang
- vi utnytter array-ordningen i startlistene
- vi kan enkelt lage nye ordninger av hoppere fra gamle, for eksempel å snu resultatlisten fra 1. omgang og la den være startlisten i 2. omgang

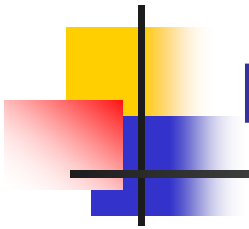
Datastruktur i class Konkurranse

```
class Konkurranse {
    final int MAX_ANTALL = 60;
    Skihopper[] deltager = new Skihopper[MAX_ANTALL];
    int antallHoppere = 0; // Brukes som indeks i deltager      ...

    void kommandoløkke() {...}

    void registrerDeltager(){
       deltager[antallHoppere++] = new Skihopper();
    }
    ...
}
```

Delegering av ansvar: Skihopper-objektet er selv ansvarlig for å innhente opplysinger om seg selv fra brukeren!

Konstruktøren i Skihopper-klassen henter info

```
class Skihopper {
    private static final int UDEFINERT = -1;
    String navn,idrettslag;
    int startnr = UDEFINERT;

    Skihopper(){
        In tast = new In();
        System.out.println("*** NY DELTAGER ***");
        System.out.print("  Navn: ");
        navn = tast.inLine();
        System.out.print("  Klubb: ");
        idrettslag = tast.inLine();
    }
```



Registrering av ny deltager

Deltagere skal registreres med navn og klubb. Det skal foretas en trekning ut fra tilfeldig genererte tall, hvoretter deltagere gis et startnummer.

```
Valg (9 for Meny): 1
*** NY DELTAGER ***
  Navn: Arne Maus
  Klubb: Institutt for informatikk

Valg (9 for Meny): 1
*** NY DELTAGER ***
  Navn: Arne Skodvin
  Klubb: Pedagogisk forskningsinstitutt
```

case 3: listDeltagere() og metoden toString()

```
    void listDeltagere(){
        for(int i=0; i<antallHoppere; i++)
            System.out.println( deltager[i] );
    }


class Skihopper {
    ...
    public String toString(){
        String s = "";
        if( startnr != UDEFINERT ) s += startnr + " ";
        s += navn +" "+ idrettslag;
        return s;
    }
}
```

Her skriver vi ut et Skihopper-objekt direkte i utskriftssetningen.

Dette krever at vi har skrevet en metode i class Skihopper med signaturen:

public String toString()

Java-systemet vil automatisk utføre denne metoden når objektet skal skrives ut!

case 2: trekning ();

```
void trekning(){
        Skihopper[] temp = new Skihopper[antallHoppere];
        for( int i=0; i<antallHoppere; i++ )
            temp[i] = deltager[i];
        deltager = temp;
        stokk();
        for( int i=0; i<antallHoppere; i++ )
            deltager[i].startnr = i+1;
        System.out.println("Trekning ferdig (det kan ikke registreres nye deltagere) ");
        trukket = true;
    }
```

Her opprettes en full array av alle deltagere

Dette gjøres for å slippe å overføre parameteren antallHoppere

NB! Vi slipper dette hvis vi bruker Arraylist isteden!

Vi oppdaterer så startnumre i Skihopper-objektene (bør ideelt gjøres med via en ”set-metode”!)

void stokk(...) og Random tall-generator

```
import java.util.Random;
Random tall = new Random();


   void stokk( ) {
       int j,k;
       Skihopper temp;
       for( int i=0; i<100; i++ ){
           j = tall.nextInt(deltager.length);
           k = tall.nextInt(deltager.length);
           temp=deltager[j];
          deltager[j]=deltager[k];
          deltager[k]=temp;
       }
   }
```

Random-klassen har en rekke funksjoner for å generere tilfeldige data på ulike format.

nextInt(int max) gir en int k i intervallet 0<= k < max

Her kunne vi unngått å overføre deltager-arrayen som parameter

Tall-generatoren kan brukes til å lage testdata

```
String[] fornavn = { "Odin", "Alf", "Even", "Ulf", "Elg", "Tor", "Rolf" };
String[] suffiks = { "snes", "sen", "svik", "shaug", "sdal", "sbakken", "sli",
"sletten" };
String[] klubb = { "TIL", "HIL", "FIL", "BIL", "MIL", "KIL" };
```

Vi lager en ny konstruktør for class Skihopper som kan ta inn data utenfra.

```
Skihopper genererNyHopper(){
    String navn = fornavn[tall.nextInt(fornavn.length)] +" "+
                fornavn[tall.nextInt(fornavn.length)] +
suffiks[tall.nextInt(suffiks.length)];
    String idrettslag = klubb[tall.nextInt(klubb.length)];
    return new Skihopper( navn, idrettslag );
}
```

Opprettelse av Omgang-objekter

```
case 4: if( !trukket ) break;
        if( førsteOmgang == null ) førsteOmgang = new Omgang( deltager,
    true );
         førsteOmgang.kommandoløkke(); break;
case 5: if( førsteOmgang == null ) break;
        if( andreOmgang == null )
          andreOmgang = new Omgang( reverser(førsteOmgang.rekkeflg),
    false );
        andreOmgang.kommandoløkke(); break;


class Omgang {
...
    Omgang( Skihopper[] startliste, boolean førsteomgang){
       // initialisering
}
```

Omgang-klassen: Konstruktør

```
class Omgang {
    Skihopper[] startliste;
    Skihopper[] rekkeflg;
    int antall;  // antall som har hoppet
    boolean førsteomgang;   // overføres til konstruktøren

    Omgang( Skihopper[] startliste, boolean førsteomgang){
        this.startliste = startliste;
        this.førsteomgang = førsteomgang;
        rekkeflg = new Skihopper[startliste.length];
    }
```

Merk at vi via startlisten får tilgang til alle skihopperne som skal starte i denne omgangen.

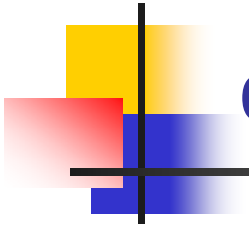
Vi overfører altså hele datastrukturen med Skihopper-objekter.

Meny for hver omgang

I hver omgang skal vi foreta registrering av nye hopp, holde orden på hvem som er neste hopper, samt resultatlisten.

```
*** MENY 1. OMGANG ***
0. Tilbake til hovedmenyen
1. Registrer nytt hopp
2. List gjenstående hoppere
3. Resultatliste
4. Simuler resten av omgangen
```

Hoppene skal registreres med lengde og 5 stilkarakterer. Startlisten for 2. omgang lages ved å snu resultatlisten for 1. omgang opp ned.



class Skihopper og class Hopp

```
class Skihopper {
    String navn,idrettslag;
    int startnr;
    Hopp førstehopp, andrehopp;
    ...
}

class Hopp {
    double lengde;
    double[] karakter;
    double poeng;
    ...
}
```

- startnr må angis etter at objektet er opprettet
- Delegering av ansvar tilsier at vi bør legge rutinen for utskrift av data om skihopperen til skihopperen selv
- Vi må støtte separat utskrift fra både 1. og 2. omgang.
- Dette kan vi oppnå enten ved å overføre en parameter som sier hvilken omgang vi ønsker utskrift for eller ved separate metoder som kalles fra hver omgang.

Kommandoløkken i Omgang-objektet

```
skrivMeny();
    do {
        System.out.print("\nValg (9 for meny): ");
        valg = tast.inInt();
        switch(valg){
            case 0: System.out.println(); break;
            case 1: nesteHopp(); break;
            case 2: skrivGjenstående(); break;
            case 3: skrivResultat(); break;
            case 4: simulerOmgang(); break;
            case 9: skrivMeny(); break;
            default: System.out.println("Du tastet feil");
        }
    } while (!(valg == 0));
}
```

Registrering av nytt hopp

```
void nesteHopp(){
    if( antall >= startliste.length ) return;
    startliste[antall].nyttHopp(førsteomgang);
    oppdaterListe(); // sett sortert inn i resultatlisten
 }
```

- Innlesning av data om hoppet og beregning av poengsum delegeres til Skihopperen og derfra videre til Hopp-klassen
- Når vi registrerer et nytt hopp, øker vi en lokal tellevariabel ”antall” med én og setter en referanse til den aktive hopperen inn i en array ”rekkeflg” slik at den holdes sortert på hoppernes poengsum.
- For å sikre at Skihopper-objektet returnerer riktig poengsum, overføres den boolske variabelen ”førsteomgang”.

Innlesing av hopp-data i Hopp-klassen

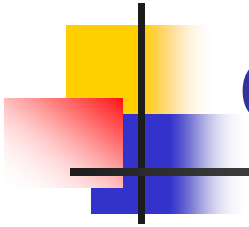
```
Hopp() {
        In tast = new In();
        System.out.print("  Lengde: ");
        lengde = tast.inDouble();
        karakter = new double[5];
        for(int i=0; i<5; i++){
            System.out.print("  Dommer " + (i+1) + ": ");
            karakter[i] = tast.inDouble();
        }
        poeng = Poengberegning.poengsum(lengde, karakter);
    }
```

Poengberegning kan legges i en egen klasse

```
class Poengberegning {

    private static final int TABELLPUNKT = 120;
    private static final double FAKTOR = 1.8;
    private static final int STARTPOENG = 60;

    static double poengsum( double lengde, double[] karakterer ){
        double tillegg = (lengde - TABELLPUNKT)*FAKTOR;
        double lengdepoeng = STARTPOENG + tillegg;
        double sum = lengdepoeng + stilsum( karakterer );
        return sum;
    }
...
}
```

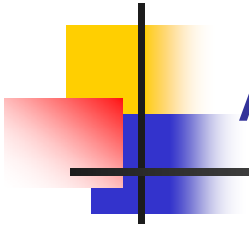
Begegning av stilkarakterer: kutt laveste og høyeste stilkarakter og summer

```
private static double stilsum( double[] karakterer ){
      int ant = karakterer.length;
      double[] sortert = new double[ant];
      for (int i=0; i<ant; i++){
          sortert[i] = karakterer[i];
          int j=i;
          while ( j>0  ) {
              if (sortert[j]<sortert[j-1]){
                  double temp = sortert[j-1]; sortert[j-1] = sortert[j]; sortert[j] = temp;}
              j--;
          }
      }
      double sum = 0;
      for (int i=1; i<ant-1; i++) sum = sum + sortert[i];
      return sum;
  }
```



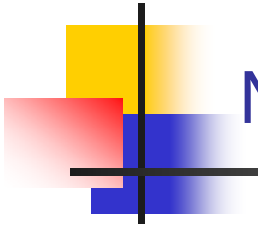
Oppsummering

- I hopprennet utspenner vi også en liten verden, en gruppe av objekter med dedikerte oppgaver.
- Vi finne klasseinndelingen til hopp-programmet fra substantivene i oppgaveteksten.
- Når vi gikk gjennom programmet gikk vi gjennom klassene ”ovenfra-ned” og skisserte en klasse først når vi fikk behov for et objekt av den.
- Vi definerte navn på metoder før vi visste hvordan de kunne kodes, og så på koden i rekkefølgen ”utenfra-inn”: vi sporet koden i omtrent samme rekkefølge som programmet eksekverer.
- Når dere selv skal skrive programmer er det ofte lurt å skrive kode i samme rekkefølge!
- Dette gir dere en metode dere kan følge som tar dere fra oppgavetekst til program i små, naturlige steg.



Å tenke objekt-orientert

- Vi tenker objekt-orientert i valg av datastruktur når vi lar objektenes funksjon være det vi primært har for øye.
- Vi må hele tiden spørre: hva kan dette objektet gjøre for meg?
- Når vi velger datastruktur, så tenk på at alle data også kan betraktes som objekter. Hvilke tjenester tilbyr de, hva kan de gjøre for meg?
- Dette passer riktignok ikke perfekt inn i enhver situasjon, snarere er det pedagogiske tommelfingerregler, men de kan likevel hjelpe deg til å gjøre gode valg av datastruktur og klasser underveis.



Noen andre funksjoner vi kan implementere

- Hvordan har en bestemt hopper gjort det i hver omgang? Deler av oppgaven kan delegeres til Skihopper-objektet og Omgang-objektet
- Hvordan har de ulike dommerne dømt? Vi må scanne gjennom alle skihopperne og finne alle hopp
- Utvidelse av programmet til å administrere flere ulike årsklasser (for eksempel ”Gutter 14. år”): Opprett et nytt Konkurranse-objekt for hver årsklasse
- Tilpassing til kombinert (2 beste av 3 omganger er tellende): Poengrutinene i class Skihopper må endres/utvides