Uke 12 - Programeksempel: System for å administrere hopprenn

15 november 2005,
Arild Waaler
Inst. for informatikk, UiO

Forelesningen i dag:

- Gå gjennom et programeksempel i detalj
- Repetere objekt-orientering
- Repetere noen vanlige programmerings-konstruksjoner

Målet for i dag og neste gang er først og fremst å hjelpe de som synes de ikke har så godt grep om programmeringen!

- Vi skal først gå gjennom problemet og designe en løsning ”ovenfra og ned”
- Så spesifiserer vi noen sentrale metoder ”utenfra og inn”

Hovedmenyen

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
```


Meny for hver omgang

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: Arild Waaler
  Klubb: UiO

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

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.

Det er 4 viktige substantiver i beskrivelsen

- Konkurranse
- Skihopper
- Omgang
- Hopp

Hver av disse danner naturlige klasser i programmet.

Fra menyene ser vi videre at

- en konkurranse inneholder 2 omganger
- en konkurranse registrerer nye skihoppere
- en omgang registrerer nye hopp

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

Hva med en Hashmap?

- En (eller flere) Hashmap vil tillate effektiv oppslag på skihoppere. Med den foreslåtte array-strukturen vil informasjonen om hver hopper ligge lagret i tabeller som evt. må søkes gjennom
- Men merk at den opprinnelige startlisten støtter de samme operasjonene som en Hashmap når nøkkelen er startnummer.
- Hvis nøkkelen er navn, kan vi enkelt finne frem til hopperen ved å scanne gjennom startlisten. Siden startlisten uansett blir ganske kort, er det ikke nødvendig med en Hashmap for et slikt søk.

class Konkurranse og class Omgang

```
class Konkurranse {
    Skihopper[] deltager;
    Omgang førsteOmgang, andreOmgang;
    ...
    void kommandoløkke() {...}
}

class Omgang {
    Skihopper[] startliste;
    Skihopper[] rekkeflg;
    boolean førsteomgang;
    ...
    void kommandoløkke() {...}
}
```

Poengutskrift fra første og annen omgang behandles forskjellig, og en boolsk variabel viser hvilken omgang som er gjeldende.

startliste og førsteomgang angis i konstruktøren

rekkeflg lagrer suksessivt resultatlisten sortert på poeng

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 (eller informasjonen som skal skrives ut)

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.

”Top down”: kommandoløkken i Konkurranse

```
skrivMeny();
do { valg = tast.inInt()
        switch(valg){
          case 0: break;
          case 1: registrerDeltager(); break;
          case 2: trekning(); break;
          case 3: listDeltagere(); break;
          case 4: førsteOmgang = new Omgang( deltager, true );
                  førsteOmgang.kommandoløkke(tast); break;
          case 5: andreOmgang = new Omgang( reverser(førsteOmgang.rekkeflg), false );
                  andreOmgang.kommandoløkke(tast); break;
          case 6: autogenerer(); break;
          case 9: skrivMeny(); break;
        }
   } while (!(valg == 0));
```

Vi må legge inn noen tester før vi kaller metoder, for eksempel:

- vi skal ikke kunne opprette andreOmgang før førsteOmgang er ferdig
- vi må trekke startnummer før første omgang kan starte

case 1: registrerDeltager();

```
    int antallHoppere = 0;
    ...
    void registrerDeltager(){
       deltager[antallHoppere++] = new Skihopper(tast);
    }

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

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

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 (en forenkling vi kan ønske å endre på senere ved utvidelse av programmet!)

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" };


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 );
}
```

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

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!

Opprettelse av Omgang-objekter

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

class Omgang {
...
    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.

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;
      Skihopper aktiv = startliste[antall];
      System.out.println("  Hopper nr. "+ (antall+1) +": "+ aktiv.navn);
      Hopp h = new Hopp();
      registrerHopp(h);
   }
```

- Innlesning av data om hoppet og beregning av poengsum delegeres 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”.
- registrerHopp(...) skilles ut som egen metode fordi simulatormetoden også vil trenge denne funksjonen.

Beregning av poengsum (boka seksjon 5.9)

Oppgave 9

Poengberegningen i skihopp baserer seg dels på hopplengde, dels på stilkarakter. Poengsummen er summen av lengdepoeng og stilpoeng.

I unnarennet fins et tabellpunkt (f.eks. 120 meter) og lengdepoengene beregnes i forhold til denne lengden. Et hopp på 120 meter gir 60 lengdepoeng. Dersom hoppet er lengre, gis et tillegg som er antall meter over 120 ganget med en faktor (meterverdi), f.eks. 1.8. Et hopp på 123 meter gir dermed 60 + (123 - 120)*1.8 = 65.4 lengdepoeng. Hoppes det under 120 meter trekkes det tilsvarende fra. Nøyaktigheten i lengdemåling går til nærmeste halvmeter.

Stilkarakterene gis som fem tall mellom 0 og 20, med en nøyaktighet på et halvt poeng. Det høyeste og laveste tallet strykes, og summen av de tre gjenværende utgjør stilpoengene. Stilkarakterene 17.0, 18.0, 18.0, 18.5 og 19.0 gir dermed 54.5 stilpoeng idet 17.0 og 19.0 blir strøket.

Lag et program som leser lengde og stilkarakterer fra terminal og beregner poengsummen ut fra et tabellpunkt på 120 meter og en meterverdi på 1.8.

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;
   }
```


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);
    }
```


Åpenbare mangler ved programmet

- Ikke robust: det sjekkes for eksempel ikke på om vi prøver å legge inn flere deltagere enn 60 (programmet vil da terminere med meldingen ”java.lang.ArrayIndexOutOfBoundsException”
- Ikke fleksibelt: når vi har tastet inn data, kan de ikke endres igjen etterpå. Hva hvis en hopper ikke stiller til start eller får hoppe om igjen? Eller vi taster feil stilkarakter?
- Ingen data lagres til fil
- Lite brukervennlig og få funksjoner
- Ved å la poeng være av double-type, får vi lett avrundingsfeil, slik at to som burde hatt samme poengsum ikke får det. Vi får lett mange uønskede desimaler i utskriften

Alt dette kan enkelt rettes på, men antall kodelinjer vil øke.
Men objektstrukturen sørger for at programmet meget lett lar seg utvide og forbedre uten at man må redesigne programstrukturen.

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