Oblig4 - obligatorisk oppgave nr. 4 (av 4) i INF1000

Leveringsfrist

Innleveringsfristen er onsdag 15. november kl 16.00. Viktig: se side 4 for detaljerte leveringskrav.

Formål

Formålet med denne oppgaven er å gi trening i hele pensum og i å lage større programmer. Løsningen du lager skal være objektorientert. Det innebærer blant annet at datastrukturen (den delen av programmet som skal "holde på dataene") skal gjøre bruk av objekter av klasser som du selv definerer. For mer detaljert informasjon om hvordan du kan gå fram for å løse oppgaven, se avsnittet *Tips og forutsetninger som kan gjøres.*

Oppgave

Meteorologisk institutt har en rekke værstasjoner rundt i Norge. Ved hver stasjon gjøres det daglig en rekke værmålinger, og resultatene samles på store datafiler for senere analyser. Hensikten med denne oppgaven er å lage et program som kan lese slike meteorologiske data fra fil, legge dataene inn i en fornuftig datastruktur, gjøre noen enkle beregninger og skrive ut resultatene på skjerm. Programmet skal være kommandostyrt.

Programmet skal gjøre bruk av data som ligger på kursets nettsider. Kopier de fire datafilene på følgende område over til ditt eget filområde: http://www.ifi.uio.no/~inf1000/obliger/DataOblig4/

Programmet skal holde rede på følgende opplysninger om hver værstasjon: stasjonsnummer, stasjonsnavn, høyde (over havet), kommune og fylke. Disse opplysningene må programmet lese fra filen **`Stasjonsdata-1.txt`**. Hver linje i denne filen inneholder opplysninger om en enkelt værstasjon. Hver opplysning er et ord eller et heltall, og opplysningene er atskilt med en eller flere blanke tegn. Her er et eksempel på en linje i filen (dette er værstasjonen på Gardermoen flyplass):

```
4780  GARDERMOEN    202  ULLENSAKER       AKERSHUS
```

Her står det altså at det er en værstasjon med stasjonsnummer 4780, stasjonsnavn GARDERMOEN, høyde over havet 202 meter, og at stasjonen ligger i Ullensaker kommune i Akershus fylke. Både stasjonsnummeret og stasjonsnavnet er entydige identifikatorer for en stasjon, dvs to forskjellige stasjoner har alltid ulikt stasjonsnummer og ulikt stasjonsnavn.

I tillegg til opplysningene om værstasjoner skal programmet holde rede på daglige værmålinger som er gjort ved hver av stasjonene i tidsrommet 01.01.2003 – 30.06.2003 (altså de seks første måneder av 2003). De daglige målingene vi er interessert i er maksimal vindhastighet, nedbørsmengde, minimumstemperatur og maksimumstemperatur. Disse måledataene må programmet lese fra filen **`Meteorologidata-1.txt`**. Måledataene ligger stasjonsvis på filen, dvs først kommer alle data for en stasjon, deretter for en annen stasjon, osv. Hver linje i filen inneholder følgende opplysninger for én bestemt stasjon og én bestemt dag: stasjonsnummer, dag-i-måned, måned, maksimal vindhastighet, nedbørsmengde, minimumstemperatur og maksimumstemperatur. Her er to av linjene i filen:

```
4780 14 01 10.8  0.3  -2.4   6.2
4780 15 01  8.7  0.0   1.4   5.0
```

Stasjonsnummeret er 4780 i begge linjer (fra den andre filen vet vi da at dette er stasjonen med navn GARDERMOEN). Ut fra den første linjen vet vi altså at det ved GARDERMOEN værstasjon den 14. januar var en maksimal vindhastighet på 10.8 m/sek, en nedbørsmengde på 0.3 mm, en minimumstemperatur på -2.4 C og en maksimumstemperatur på 6.2 C. Tilsvarende for neste linje.

For noen av datafeltene i filen **`Meteorologidata-1.txt`** mangler det observasjoner, og da står verdien -99 der det skulle stått en måling. Du må sørge for at forekomstene av -99 ikke tas med i beregninger, f.eks. når du skal regne ut et gjennomsnitt.

Når du er sikker på at programmet ditt virker, skal du erstatte filene ovenfor med de to filene **`Stasjonsdata-2.txt`** og **`Meteorologidata-2.txt`**. Disse er bygget opp på akkurat samme måte som de to første filene, men de inneholder data om mange flere værstasjoner, og du anbefales derfor å vente med å se på disse til du føler deg trygg på at programmet fungerer.

Når programmet starter opp, skal det lese filen med stasjonsdata og filen med værdata, og dataene legges inn i den datastrukturen du har definert. De to datafilene bør du på forhånd ha kopiert over til ditt eget område, slik at de ligger på samme filområde som du kjører Java-programmet fra. Programmet skal deretter være kommandostyrt, dvs det skal kunne ta imot en kommando fra brukeren, utføre kommandoen, ta imot ny kommando, osv, helt til brukeren ønsker å avslutte. Mer konkret skal programmet oppføre seg slik sett fra brukerens side:

- Det skriver ut på skjerm hvilke kommandoer brukeren kan gi.
- Deretter ber programmet om og leser inn en kommando fra brukeren.
- Programmet utfører den valgte kommandoen.

Programmet skal gjenta de tre trinnene ovenfor helt til brukeren gir kommando om å avslutte (se nedenfor). Brukeren skal kunne gi følgende kommandoer:

- **Lag stasjonsliste.** Programmet skal da skrive ut på skjermen en sortert liste over alle stasjoner. Listen skal inneholde en linje for hver stasjon og denne linjen skal inneholde stasjonsnavnet, stasjonsnummeret, kommunen, fylket og høyden til stasjonen (i den rekkefølgen). Listen skal være sortert med hensyn på stasjonsnavnet.

- **Lag stasjonsliste for fylke.** Programmet skal da be om og lese inn navnet på et fylke (f.eks. AKERSHUS) og deretter skrive ut på skjermen en sortert liste over alle stasjonene som ligger i dette fylket. Listen skal inneholde en linje for hver stasjon, og denne linjen skal inneholde stasjonsnummeret, stasjonsnavnet, kommunen og høyden til stasjonen. Listen skal være sortert med hensyn på stasjonsnavnet.

- **Lag månedsoversikt.** Programmet skal da be om å få oppgitt enten nummeret eller navnet til en stasjon (brukeren kan da f.eks. svare ”4780” eller ”GARDERMOEN”). Svaret leses inn og deretter skal det skrives ut en månedsvis oppsummering av været ved denne stasjonen. Mer presist skal du for hver måned skrive ut en linje på skjermen som angir måned, gjennomsnittlig nedbør, gjennomsnittlig minimumstemperatur, gjennomsnittlig maksimumstemperatur og høyeste målte maksimumstemperatur den måneden. Listen kan f.eks. starte slik:

  ```
  Januar      0.1   -5.2   2.4   9.5
  Februar     0.3   -2.1   6.7   13.2
  .......
  .......
  ```

- **Sammenlikne nedbør kyst/innland.** En kyststasjon er definert som en stasjon med høyde < 60, og en innlandsstasjon er en stasjon med høyde ≥ 60. Programmet skal, måned for måned, sammenlikne været ved kyststasjoner med været ved innlandsstasjoner. Mer presist skal du for hver måned beregne to verdier som vi kan kalle nedbørKyst og nedbørInnland, og skrive ut en linje på skjermen som angir måned, nedbørKyst, nedbørInnland og hvor det er mest nedbør (KYST eller INNLAND). Verdien til nedbørKyst skal du finne i to steg: (1) beregne for hver kyststasjon hva gjennomsnittlig nedbør var den gitte måneden; og (2) beregne gjennomsnittet av de verdiene du fant i punkt (1) og kalle dette gjennomsnittet for nedbørKyst. Verdien til nedbørInnland finner du på helt tilsvarende måte. Husk å bare telle med de dagene hvor det er gjort observasjoner, dvs ikke ta med de dagene hvor filen inneholdt verdien -99 i feltet for nedbør for den gitte stasjonen.

- **Finn antall uværsdager.** Programmet skal da spørre brukeren om å få oppgitt nummeret eller navnet til en stasjon og en måned (et heltall mellom 1 og 6). Så skal programmet gå gjennom alle observasjonsdata for den valgte værstasjonen og den valgte måneden, og telle opp antall uværsdager. En uværsdag er en dag hvor *minst en av* følgende betingelser er oppfylt:
  - nedbørsmengden er større enn eller lik 10
  - maksimal vindhastighet er større enn eller lik 12

  Dvs. du skal finne antall dager det enten regner mye eller blåser mye – eller begge deler. Til slutt skrives stasjonsnavn, måned og antall uværsdager ut på skjermen.

- **Sammenlikne været Østlandet/Nord-Norge.** Dette punktet kan du gjøre hvis du får tid og du har sjekket at alt annet i programmet fungerer som det skal. Programmet skal først spørre brukeren om en måned (et heltall mellom 1 og 6). Deretter skal du for Østlandet og for Nord-Norge hver for seg beregne gjennomsnittlig middeltemperatur og gjennomsnittlig nedbørsmengde. Mer presist skal du for Østlandet finne to størrelser tempØst og nedbørØst, og tilsvarende for Nord-Norge. Østlandet omfatter her alle værstasjoner i fylkene Akershus, Oslo, Oppland, Buskerud og Vestfold. Nord-Norge omfatter alle værstasjoner i Nordland, Troms og Finnmark.

  Du finner tempØst i to steg: (1) beregne for hver stasjon på Østlandet hva gjennomsnittlig middeltemperatur var den gitte måneden; og (2) beregne gjennomsnittet av de verdiene du fant i punkt (1). Gjennomsnittlig middeltemperatur for en gitt stasjon finner du ved å summere middeltemperaturen over alle dager den gitte måneden og dele på antallet dager (husk å korrigere for eventuelle manglende observasjoner). Middeltemperaturen for en gitt stasjon og dag er definert som (minimumstemperatur + maksimumstemperatur) / 2.0.

  Du finner nedbørØst på tilsvarende måte i to steg: (1) beregne for hver stasjon på Østlandet hva gjennomsnittlig nedbørsmengde var den gitte måneden; og (2) beregne gjennomsnittet av de verdiene du fant i punkt (1). For en gitt stasjon finner du gjennomsnittlig nedbørsmengde ved å summere opp nedbøren over alle dager den gitte måneden og dele på antallet dager (husk også her å korrigere for eventuelle manglende observasjoner).

  For å finne de tilsvarende størrelser tempNord og nedbørNord for Nord-Norge, gjør du akkurat som ovenfor, men nå basert på alle værstasjonene i Nord-Norge.

  Skriv til slutt ut resultatene på skjermen.

- **Avslutt.** Programmet skal da avslutte.

Leveringskrav

Som for tidligere obligatoriske oppgaver må du følge de generelle krav til innleverte oppgaver ved institutt for informatikk: http://www.ifi.uio.no/studinf/skjemaer/erklaring.pdf .

Oppgaven skal utføres og leveres individuelt via Joly-systemet. Innen fristen skal du ha levert:

- Det ferdige programmet (”Oblig4.java”)

- UML-klassediagram over systemet ditt. Husk å sette navn på de forholdene du vil ha i systemet ditt. For også opp antall på begge sider av forholdene. UML-diagrammet kan leveres direkte til øvingslærer som en håndtegning, eller du kan lage tegningen med et tegneprogram og sende den på epost til øvingslærer som postscript-fil eller pdf-fil (hvis du ønsker å benytte et annet filformat må du isåfall avtale dette først med øvingslæreren). Uansett leveringsmåte må du huske å merke tegningen tydelig med ditt navn og brukernavn, og hvilken øvingsgruppe du tilhører.

- Resultatet av en testkjøring. Denne skal vise utskriften på skjermen når du starter opp programmet ditt og tester ut de forskjellige delene som du er spurt om å lage i oppgaven. Kjører du Unix, kan du bruke programmet photo for å lagre på fil alt som skrives ut på skjermen under en testkjøring. Du gir da følgende kommando i xterm-vinduet rett før du starter java-programmet ditt: *photo testOblig4.tx*t . Når du senere har avsluttet java-programmet ditt, gir du kommandoen *exit* (eller trykker control-D) for å avslutte photo-programmet. Da skal all utskrift fra testen ligge på filen *testOblig4.txt.*

Joly-systemet virker på Blindern-maskinene, og det virker som oftest også hjemmefra hvis du kjører VPN. Hvis du skal sende besvarelsen via epost, anbefales det sterkt at du benytter din epostkonto på UiO for å unngå at besvarelsen kommer bort. Hvis du har spørsmål vedrørende leveringsmåte eller annet, kontakt øvingslæreren din i god tid før innleveringsfristen. Husk at det er ditt ansvar at oppgaven kommer frem til øvingslæreren på riktig måte innen fristen.

Tips til hvordan du kan løse oppgaven

Nedenfor finner du noen tips til hvordan du kan legge opp arbeidet med å løse oblig 4 i INF1000. Tipsene er organisert i to kategorier: ting som angår programmets struktur (inkludert datastrukturen) og ting som angår programmeringen av de enkelte kommandoene.

Programmets struktur

1. Programmet du skal skrive kan for eksempel struktureres slik:

```
import easyIO.*;
import java.util.*;

class Oblig4 {
    public static void main (String[] args) {
       String s1 = "Stasjonsdata-1.txt";
       String s2 = "Meteorologidata-1.txt";
       Analyse a = new Analyse(s1, s2);
       a.ordreløkke();
   }
}

class Analyse {
   HashMap hashNavnStasjon = new HashMap();
   HashMap hashNummStasjon = new HashMap();

   Analyse(String stasjonsFilnavn, String meteoFilnavn) {
       lesStasjonsdata(stasjonFilnavn);
       lesMeteorologidata(meteoFilnavn);
   }

   void lesStasjonsdata (String fnavn) {...}
   void lesMeteorologidata (String fnavn) {...}
   void ordreløkke() {...}
   void lagListeStasjoner() {...}
   void lagListeStasjonerIFylke() {...}
   void lagMånedsoversikt() {...}
   void sammenliknKystInnland() {...}
   void finnAntallUværsdager() {...}
   void sammenliknØstNord() {...}
}

class Stasjon {...}      // Ett objekt for hver stasjon

class Maaned {...}   // Seks objekter for hver stasjon

class Dag {...}

... eventuelt flere klasser ...
```

2. Noe av det første du bør gjøre etter at du har lest og forstått oppgaven, er å tegne et UML-klassediagram slik at du ser for deg hvordan de ulike komponentene i programmet henger sammen. Ikke vent med denne jobben til programmeringen er ferdig.

3. Hver værstasjon representeres ved et eget objekt av klassen Stasjon. I denne klassen deklarerer du objektvariable for alle opplysningene om en stasjon: stasjonsnummer, stasjonsnavn, høyde, osv. I tillegg må hvert stasjonsobjekt holde rede på "sine" værdata.

4. Hver måned med værdata for en gitt stasjon representeres ved et eget objekt av klassen Maaned. I denne klassen deklarerer du de nødvendige arrayer for å holde på de fire værmålingene (maksimal vindhastighet, nedbørsmengde, minimumstemperatur og maksimumstemperatur) for hver av dagene i måneden. Siden antall dager i en måned kan variere, kan du enten la arrayene i alle slike objekter ha en fast lengde (= 31, siden dette alltid er stort nok) og ha en egen heltallsvariabel i objektene som holder rede på hvor mange dager den aktuelle måneden har, eller du kan la arrayene i slike objekter være akkurat store nok (da du må du vite antall dager i den aktuelle måneden før du oppretter arrayene).

5. Klassen Maaned kan struktureres på flere måter. En mulighet er å deklarere fire like lange double-arrayer i Maaned, en for hver av de fire værmålingene som gjøres. En annen mulighet er å lage en ny klasse Dag med fire double-variable, slik at hvert objekt av denne klassen holder på de fire værmålingene som gjøres en bestemt dag ved en bestemt stasjon. De fire arrayene i klassen Maaned kan da erstattes av en enkelt Dag-array. Begge løsninger er akseptable, og hvilken man velger blir en smakssak (sistnevnte løsning kan sies å være mer i en ”objektorientert ånd”, men på den annen side vil Dag-objektene ikke inneholde stort andre metoder enn de som skal til for å ta ut og sette inn verdier, og løsningen medfører et ekstra nivå med pekere og objekter).

6. Hvert stasjonsobjekt må holde rede på nøyaktig seks objekter av klassen Maaned (ett objekt for januar-målingene, ett objekt for februar-målingene, osv). Dette kan hensiktsmessig gjøres ved hjelp av en array

   ```
   Maaned mdata = new Maaned[6];
   ```

   i klassen Stasjon.

7. For å holde orden på alle objektene av klassen Stasjon kunne du i prinsippet benyttet en array i klassen Analyse. En annen løsning er å benytte HashMap. Begge løsninger vil føre frem, men *for å få trening i bruk av HashMap, er det dette du skal benytte i denne oppgaven.* Hvis **`st`** er en peker til et stasjonsobjekt, **`navn`** er en String-variabel med stasjonsnavnet, og **`numm`** er en String-variabel med stasjonsnummeret, så kan du legge stasjonen inn i HashMap’ene **`hashNavnStasjon`** og **`hashNummStasjon`** ved å skrive

   ```
   hashNavnStasjon.put(navn, st);
   hashNummStasjon.put(numm, st);
   ```

Disse legger stasjonsobjektet (egentlig en peker til objektet) inn i de to HashMap'ene, med henholdsvis stasjonsnavnet og stasjonsnummeret som nøkkel. Vær klar over at nøkkelen i en HashMap skal være en String (det er egentlig mer generelt enn det, men det ser vi bort fra i INF1000). Vi vil aldri være interessert i å tenke på stasjonsnummere som tall (vi vil f.eks. aldri ønske å summere to stasjonsnummere). Du anbefales derfor å lese inn stasjonsnummere fra fil og terminal *som tekststrenger* og ikke som heltall.

8. Hvorfor benytte to HashMap'er og ikke en? Årsaken er at det av og til er ønskelig å søke etter stasjonen ut fra stasjonsnummer og av og til ut fra stasjonsnavn. Med *to* HashMap'er – en hvor nøkkelen er stasjonsnavnet og en hvor nøkkelen er stasjonsnummeret, får vi både i pose og sekk. Det er de *samme* stasjonsobjektene du skal legge inn i de to HashMap'ene, men nøklene vil være ulike.

De enkelte kommandoene

1. De to første kommandoene (*lag stasjonsliste* og *lag stasjonsliste for fylke*) skal skrive ut sorterte lister over stasjoner. Siden stasjonene ligger lagret i HashMap'er kan du få tak i alle stasjonene ved å løpe gjennom en av de to HashMap'ene. Dette vil imidlertid ikke gi deg stasjonene i den rekkefølgen du ønsker (altså sortert med hensyn på stasjonsnavn). For å få laget en sortert liste, kan du gjøre følgende: (1) løp gjennom en av HashMap'ene for å få tak i alle stasjonsnavnene, og legg disse stasjonsnavnene fortløpende inn i en String-array alleNavn; (2) sorter listen av stasjonsnavn ved å utføre java.util.Arrays.sort(alleNavn); (3) løp gjennom alle navnene i alleNavn og for hvert navn slå opp i HashMap'en hashNavnStasjon for å finne tilhørende stasjon. Dette gir deg stasjonene sortert etter stasjonsnavn, og nå kan du for hver stasjon skrive ut de ønskede data på skjermen. Merk at når du skal deklarere arrayen alleNavn så er det greit å vite hvor mange stasjoner man har. Det kan derfor være greit å ha en objektvariabel i klassen Analyse som heter antallStasjoner og som du finner verdien til idet du leser inn data fra filen Stasjonsdata-1.txt.

2. For to av kommandoene (*lag månedsoversikt* og *finn antall uværsdager*) skal programmet be brukeren om et stasjonsnummer eller et stasjonsnavn. Ideen er her at brukeren selv skal få bestemme om stasjonsnummer eller stasjonsnavn skal oppgis, og så skal programmet forstå av svaret som gis om det dreier seg om et nummer eller et navn. Anta at brukerens svar ligger i String-variabelen **`svar`**. Vi vet ikke om svaret er et stasjonsnavn eller et stasjonsnummer, så vi forsøker begge HashMap'ene (i vilkårlig rekkefølge):

```
Stasjon st = null;
if (hashNummStasjon.containsKey(svar)) {
     st = (Stasjon) hashNummStasjon.get(svar);
} else if (hashNavnStasjon.containsKey(svar)) {
     st = (Stasjon) hashNavnStasjon.get(svar);
} else {
     // Feil oppstått: brukeren har svart noe som verken er
     // et gyldig stasjonsnummer eller et gyldig stasjonsnavn.
     // Gi brukeren beskjed om dette, og avslutt utførelsen
     // av kommandoen
}
```

3. Når du leser inn stasjonsnavn fra filen `Stasjonsdata-1.txt` eller `Stasjonsdata-2.txt` så er stasjonsnavnene skrevet med store bokstaver. Det betyr at når du senere skal søke etter en stasjon basert på stasjonsnavnet, så må du også da skrive stasjonsnavnet med store bokstaver (ellers vil vi ende opp i den siste else-grenen (”Feil oppstått”) i punkt 7 ovenfor. Alternativt kunne du ha konvertert brukerens svar til store bokstaver før du søker i HashMap’en **`hashNavnStasjon`** (du trenger ikke å gjøre dette i denne obligen).

4. Det neste du bør gjøre er å skrive ordreløkken. Når programmet har funnet ut hvilken ordre som skal utføres, skal ordreløkken kalle på en metode som utfører ordren. *Ordreløkken selv skal altså ikke inneholde selve programkoden som utfører kommandoen*. Årsaken til dette er at det er mye mer oversiktlig at ordreløkken delegerer oppdrag (til andre metoder) enn at alt skal foregå i selve løkken.

5. Metodene som kalles fra ordreløkken skal ikke ha parametre, og de skal heller ikke returnere noen verdi (dvs. de skal være av typen void). Årsaken er enkel: siden ordreløkken delegerer videre alt ansvar for å utføre kommandoen, så skal også kommunikasjon med bruker foregå utenfor ordreløkken (og dermed er det ikke særlig poeng i parametere eller returverdier). Den eneste kommunikasjonen med bruker som ordreløkken skal stå for, er valg av neste kommando. Eksempel: hvis brukeren ønsker å utføre kommandoen ”Finn antall uværsdager”, så skal ordreløkken kalle på metoden **`finnAntallUværsdager()`**.

6. Metodene som kalles fra ordreløkken (f.eks. **`finnAntallUværsdager()`** i eksemplet i punktet over), ihvertfall de som skal utføre beregninger, kan med fordel struktureres slik: (1) de henter inn nødvendig informasjon fra brukeren; (2) de kaller på en metode (med parametere og med returverdier) som utfører selve operasjonen og returnerer resultatet; (3) de skriver ut resultatet på skjermen. Det betyr at metoden **`finnAntallUværsdager()`** vil inneholde et kall på en metode

   ```
   int antUværsdager(String stNumm, int måned) {....}
   ```

   som du også må skrive. Nøyaktig hvordan arbeidsdelingen skal være mellom de to metodene er litt opp til deg, men forsøk å lage et naturlig skille mellom hva som er ansvarsområdet for de to metodene.

7. Skriv deretter de delene av programmet som leser de to filene. Opprett objekter av de klassene du har definert etter hvert som du leser fra filene. Alle klassene (med unntak av den som inneholder main-metoden) skal ha en konstruktør som du har skrevet selv og som initierer objektet med de data du leser inn om vedkommende objekt. *Sjekk så nøye du kan at innlesningen fra fil og oppretting av datastruktur fungerer før du går videre med resten av programmeringen* – legg gjerne inn ekstra skjermutskrift i programmet slik at du kan sjekke hva som faktisk skjer ulike steder (slik skjermutskrift må du naturligvis fjerne før du leverer oppgaven).

8. En måte å behandle manglende data på, er at alle -99’ene leses inn som om dette er virkelige data, og at metoder som regner ut gjennomsnitt ol, hver gang sjekker om det er verdier != -99 de finner og bare tar med dataverdier som != -99. Slike metoder kan også selv returnere -99 dersom det ikke finnes noen data å gjøre beregningene på (enten for eksempel for at brukeren spesifiserer en måned vi ikke har data for - f.eks. måned: 8 – august, eller at alle data for vedkommende måned mangler).