Framgangsmåte for å løse oblig 4 i INF1000

Nedenfor finner du noen tips til hvordan du kan legge opp arbeidet med å løse oblig 4 i INF1000.

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 = "Stasjoner-1.txt";
       String s2 = "Vaerdata-1.txt";
       if (args.length >= 2) {
            s1 = args[0];
            s2 = args[1];
       }
       VaerAnalyse va = new VaerAnalyse(s1, s2);
       va.ordreløkke();
    }
}

class VaerAnalyse {
    HashMap <String,Stasjon> stasjonFraNavn = new HashMap <String,Stasjon> ();
    HashMap <String,Stasjon> stasjonFraNr = new HashMap <String,Stasjon> ();

    VaerAnalyse(String stasjonsfilnavn, String vaerfilnavn) {
        lesStasjonerFraFil(stasjonsfilnavn);
        lesVaerDataFraFil(vaerfilnavn);
    }

    void lesStasjonerFraFil(String fnavn) {...}
    void lesVaerDataFraFil(String fnavn) {...}
    void ordreløkke() {...}
    void lagStasjonsliste() {...}
    void finnAntUværsdager() {...}
    void finnRegnKystInnland() {...}
    void sammenlignNordVest() {...}
}

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

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

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

2. Klassen Oblig4 inneholder main-metoden og gjør ikke stort annet enn å få programmet i gang. Når programmet skal kjøres gir du en av kommandoene

   `java Oblig4`

   eller

   `java Oblig4 <stasjonsdatafil> <værdatafil>`

I første tilfellet vil programmet benytte de to filene "Stasjoner-1.txt" og "Vaerdata-1.txt". I siste tilfellet vil programmet benytte de to filene som oppgis. Setningen **`VaerAnalyse va = new VaerAnalyse(s1, s2)`** kaller så på konstruktøren i klassen VaerAnalyse og overfører de to filnavnene til denne konstruktøren, slik at filene kan åpnes for lesing. Når du senere vil prøve å lese de større filene, gir du kommandoen

```
java Oblig4 Stasjoner-2.txt Vaerdata-2.txt
```

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 Månedsdata. I denne klassen deklarerer du de nødvendige arrayene for å holde orden 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 Månedsdata kan struktureres på flere måter. En mulighet er å deklarere fire like lange double-arrayer i Månedsdata, en for hver av de fire værmålingene som gjøres. En annen mulighet er å lage en ny klasse Dagdata 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 Månedsdata kan da erstattes av en enkelt Dagdata-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 Dagdata-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 Månedsdata (ett objekt for januar-målingene, ett objekt for februar-målingene, osv). Dette kan hensiktsmessig gjøres ved hjelp av en array **`Månedsdata mdata = new Månedsdata[6]`** i klassen Stasjon.

7. For å holde orden på alle objektene av klassen Stasjon kunne du i prinsippet benyttet en array i klassen VaerAnalyse. 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, **`stNavn`** er en String-variabel med stasjonsnavnet, og **`stNummer`** er en String-variabel med stasjonsnummeret, så kan du legge stasjonen inn i HashMap’ene **`stasjonFraNavn`** og **`stasjonFraNr`** ved å skrive

   ```
   stasjonFraNavn.put(stNavn, st);
   stasjonFraNr.put(stNummer, st);
   ```

   Disse legger stasjonsobjektet (egentlig en peker til objektet) inn i de to HashMap’ene, med henholdsvis stasjonsnavn og stasjonsnummer 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.

   For å få tak i stasjonsobjektet svarende til stasjonsnavnet **`stNavn`**, kan du skrive

   ```
   Stasjon st = stasjonFraNavn.get(stNavn);
   ```

og tilsvarende for den andre HashMap'en hvis du kjenner stasjonsnummeret. Vær klar over at hvis du forsøker å få tak i et stasjonsobjekt og oppgir et stasjonsnavn som ikke finnes i HashMap'en, så vil du få returnert verdien **`null`**, dvs. vi vil da ha **`st == null`** etter at setningen ovenfor er utført. Det kan være lurt å sjekke spesielt på dette i programmet, slik at du unngår rare feilmeldinger (NullPointerException's) senere under programeksekveringen. Tilsvarende gjelder naturligvis for den andre HashMap'en, dvs. hvis du forsøker å hente et stasjonsobjekt fra **`stasjonFraNr`** og oppgir et stasjonsnummer som ikke er registrert i HashMap'en så får du returnert verdien **`null`**.

8. Hvorfor er det 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 (selv om det ikke står eksplisitt i oppgaveteksten, kan du anta at stasjonsnavnet også entydig identifiserer en værstasjon). 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.

9. 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.

10. 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 langt mer oversiktlig at ordreløkken delegerer oppdrag (til andre metoder) enn at alt skal foregå i selve løkken.

11. Metodene som kalles fra ordreløkken skal ikke ha parametere, 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 **`finnAntUværsdager()`**.

12. Metodene som kalles fra ordreløkken (f.eks. **`finnAntUværsdager()`** i eksemplet i punktet over), i hvert fall 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 **`finnAntUværsdager()`** vil inneholde et kall på en metode
    ```
    int antUværsdager(String stasjonsNr, 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.

13. 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).

14. Lag tre hjelpemetoder i den klassen som representerer måledata for en måned for en stasjon. Den første regner ut gjennomsnittlig nedbør per dag den måneden; den andre metoden regner ut gjennomsnittlig temperatur (middeltemperatur) for måneden ut fra max- og min-temperaturene for hver dag, og den siste metoden teller opp antall uværsdager.

15. 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). Husk at du samtidig må telle opp hvor mange dager du har virkelige observasjoner for, slik at gjennomsnittet blir riktig.

16. Når du skal regne ut f.eks. gjennomsnittlig nedbør over alle måneder for en bestemt stasjon, så kan du gå i løkke gjennom de ulike månedene og summere sammen gjennomsnittlig nedbør for hver av månedene. Til slutt deler du denne summen på antall måneder. (Selv om resultatet du da får ikke er eksakt likt det virkelige gjennomsnittet av alle målingene over alle måneder, vil forskjellen typisk være svært liten).

17. Når du skal finne ut om det regner mest med kysten kan et hint være å først lage en metode som regner ut gjennomsnittlig daglig nedbørsmengde for en bestemt stasjon (dette kan regnes ut ved å summere nedbøren over alle dager for denne stasjonen, og dele på antallet dager – men husk å bare telle med de dagene hvor det er gjort observasjoner og ikke de hvor det mangler verdi for nedbøren). Lag så to double-variable sumKyst og sumInnland og initialiser dem til 0. Nå kan du finne de to verdiene du skal beregne ved å løpe i en løkke gjennom alle stasjoner og addere stasjonenes gjennomsnittlige nedbør til sumKyst (hvis det er en kyststasjon) eller til sumInnland (hvis det er en innlandsstasjon). Til slutt deler du hver av de to summene på henholdsvis antall kyststasjoner og antall innlandsstasjoner (disse antallene kan du beregne i løkken ovenfor).

18. Når du skal sammenlikne Nord-Norge og Vestlandet kan det være lurt å først laget en metode som for en bestemt stasjon regner ut gjennomsnittlig daglig nedbørsmengde (dette har du allerede gjort i punktet over) og en metode som regner ut gjennomsnittlig middeltemperatur (dette kan du regne ut ved å summere middeltemperaturene over alle dager for denne stasjonen og dele på antallet dager – men husk også her på å telle med kun de dager hvor det er gjort målinger). Lag så fire double-variable sumNedborNord, sumNedborVest, sumTempNord og sumTempVest og initialiser dem til 0. Da kan du finne de fire verdiene du skal beregne ved å løpe i løkke gjennom alle stasjoner og addere stasjonens gjennomsnittlige nedbør og gjennomsnittlige middeltemperatur til henholdsvis sumNedborNord og sumTempNord (hvis det er en stasjon i Nord-Norge) eller til sumNedborVest og sumTempVest (hvis det er en stasjon på Vestlandet). Til slutt deler du sumNedborNord og sumTempNord på antall stasjoner i Nord-Norge, og sumNedborVest og sumTempVest på antall stasjoner på Vestlandet (disse antallene kan du beregne i løkken ovenfor).

19. Vår løsning ble på ca. 420 linjer java-kode, men din løsning kan godt bli betydelig kortere eller lengre uten at den er noe dårligere for det.

Lykke til!