Datalogi C
Henning Christiansen

Opgave til løsning ved øvelserne 19. september 2003.

Opgavens eksempel er en simpel form for datamining på tekstfiler baseret på optælling af sekvenser af ord. Formålet med opgaven er at

- genopfriske grundelementer af Java,
- se på klasser som redskaber til indpakning af sammenhørende datastrukturer og algoritmer,
- overvejelser om effektivitet.
- overvejelser om hvor nemt eller besværligt det er at udvide et eksisterende program.
- kritisk diskussion af Javas typebegreb m.v.

OBS: Hvis man har tidnød og/eller er nødt til at genopfriske de basale elementer af Java, kan det anbefales at koncentrerer sig om spørgsmål 1 og går det igennem i detaljer.

Hvad der givet på forhånd:

Indlæsning fra filer er meget besværligt i Java og ikke særligt interessant at beskæftige sig med. Derfor udleveres til opgavens løsning en klasse EasyFile.java (hent den på kursets hjemmeside). En EasyFile kan opfattes som bestående af strenge (svarende til de enkelt ord) som man så kan hente en ad gangen.

Hvis vi nu har en tekstfil med f.eks. navnet "duckling" (uden extension), så kan du benytte klassens metoder på følgende vis:

EasyFile F = new EasyFile("duckling");
Du opretter en ny EasyFile hvis indhold svarer til "duckling"

F.openEasyFileForRead();
Skal kaldes før du kan læse fra F.

F.readStringEasy()
Returnerer et String-objekt svarende til næste ulæste ord.
F.eks. String S = F.readStringEasy()

F.endOfEasyFile()
Kan du bruge til at teste om du har læst alle strenge på F.

NB: det er ikke specifiseret, hvad der sker, hvis du kalder readStringEasy() hvis endOfEasyFile() er sand.

Klassens main metode (som kun er til testformål) illustrerer metoderne:

```
public static void main(String [] args) {
  EasyFile F = new EasyFile("duckling");
  F.openEasyFileForRead();
```

```
  while( ! F.endOfEasyFile())
     System.out.println( F.readStringEasy()); }
```

Den skriver samtlige strenge fra den angivne fil ud på skærmen.

Iøvrigt: På kursets hjemmeside finder du filen "duckling" (uden extension) som indeholder en flad tekstfil med "Den grimme ælling" i engelsk oversættelse, kun med små bogstaver og alle punktuationstegn fjernet (så undgår vi æ-ø-å-problemer m.v.).

Spørgsmål 1

Der skal designes og implementeres en klasse til at repræsentere ord; kald den class Word. Det gælder her, at vi vil undgå at repræsentere hver forekomst af en tekststreng mange gange. I stedet repræsenteres hvert ord ved hjælp af et *heltal*: Derved bliver sammenligning af ord langt mere effektivt: I stedet for at sammenligne strenge hver eneste gang, kan man blot sammenligne to tal for at se om to Word-objekter er ar betragte som ens.

Anbefalet strategi: Klassen word har en hukommelse af de strenge, vi har set indtil nu. Hvert ord tildeles en unikt nummer *n*. Ud fra *n* kan vi finde tilhørende tekststreng

Eksempel:

```
new Word("sildepostej")
```

Der oprettes et objekt; hvis "sildepostej" er registreret allerede med nr. 17, så placeres 17 i det nye objekt; ellers registreres "sildepostej" ud for det næste ubrugte nummer, f.eks. 28 (og dette lægges så i det nye Word-objekt).

For nemheds skyld kan vi nøjes med at repræsenterer hukommelsen ved et tilpas langt array af strenge (og glem alt om check for overløb). Når vi skal oprette et nyt ord, søger vi blot igennem fra en ende af (og i denne proces slipper vi så ikke for at sammenligne strenge).

Implementér klassen Word med konstruktor, equals- og toString-metoder efter de principper, som er beskrevet i bogen. Tilføj også en metode

```
public boolean wordEquals(Word rhs)
```

Forskellen ligger i at equals-metodens argument skal angives som et generelt Object, men for wordEquals tillader vi kun Word-objekter. Overvej hvor stor forskel der er i effektivitet på de to metoder, du når frem til. (NB: Ingen grund til at lave kørsler med tidsmålinger!)

Overvej i øvrigt tidsforbruget for et kald af "new Word(...)": Hvis det tager i snit 1 ms med 1000 kendte ord i hukommelsen, hvor lang tid tager det så med 2000 ord? Og med 10.000 ord?

Spørgsmål 2

Klassen word skal bruges i et program, der optæller antallet af gange ord eller sekvenser af ord forekommer i en tekst. Vi sætter en grænse på sekvenser op til 4 ord, altså, vi er interesserede i at tælle antal gange en sekvens af længde 1 (svarende til enkelt ord), og af længde 2, 3 og 4 forekommer. For eksemplet "a b c d b c d a a" svarer det til følgende:

a: 3	c: 2	d b c d: 1
a b: 1	c d: 2	b c d a: 1
a b c: 1	c d b: 1	c d a: 1
a b c d: 1	c d b c: 1	c d a a: 1
b: 2	d: 2	d a: 1
b c: 2	d b: 1	d a a: 1
b c d: 2	d b c: 1	a a: 1
b c d b: 2		

Skriv klasser til at repræsentere sekvenser og tælleværk.

For sekvenserne skal der sættes toString og equals-metoder på; overvej som i spørgsmål et at sætte en mere effektiv sammenligningsmetode end equals på. [NB: der er mange forskellige måder at sætte repeæsentere sekvenser på, og mange designvalg].

Til tælleværket kan man f.eks. benytte to arrays, et med objekter svarende til sekvenser og et med antallet af gange den givne streng forekommer. For eksempel, hvis det ene array har sekvensen "en"-"blå"-"hest" i celle 17, så vil man kunne finde antallet af gange den er forekommet i det andet arrays celle 17.

Når du får programmet til at køre, kan du overveje kriterier for ikke at skrive alle sekvenser ud, men kun de som er "interessante" at få opysninger om.

Spørgsmål 3

Indenfor data mining på tekster som illustreret her udskiller man ofte en gruppe ord kaldet stopord (en lidt misvisende betegnelse). Stopord er ord, som i sig selv ikke bærer meget betydning, men blot klistrer andre ord sammen. F.eks. "in", "on", "an".

Overvej hvor besværligt det måtte være at indarbejde begrebet stopord i programmet, og hvordan man kan ændre programmets opførsel, så sekvenser som starter eller slutter med et stopord ikke tælles med.

Der forventes ikke at være tid til at implementere dette, men for de som måtte ønske det, kan en fil "stopwords" hentes fra kursets hjemmeside.