Datalogi C
Henning Christiansen 1/11-2004

Opgave til stoffet gennemgået ved forelæsningen 2/11-2004

Der arbejdes med med opgave 1 og 2 til øvelserne 5/11; *Opgave 2 minus spørgsmål 2.1 er efterfølgende afleveringsopgave med frist 16. november* (med samme krav som til de forrige afleveringsopgaver).

Opgave 1
Betragt de hægtede lister, som er beskrevet i lærebogens afsnit 17.1. Beskriv tilstanden af L og de iterators, som indføres, efter hver sætning:

```
LinkedList L = new LinkedList();
LinkedListIterator iter = L.zeroth();
L.insert(a,iter);
L.insert(b,iter);
iter.advance();
L.insert(c,iter);
LinkedListIterator anotherIter = L.first();
anotherIter.advance();  anotherIter.advance();
L.insert(d,anotherIter);
iter.advance();
L.insert(e,iter);
```

Opgave 2
A. Jensen har et lille firma som påtager sig kopieringsopgaver. Man afleverer et antal originale sider plus bestilling af hvor mange kopier man vil have. A. Jensen kender sin kopimaskine godt, så han hurtigt kan regne ud, hvor mange minutter, hver opgave vil tage. Og det er en god kopimaskine der aldrig går istykker, og den har en elektronisk dims, som adviserer papirleverandøren som altid er på pletten og fylder papir i, så kopimaskinen aldrig løber tør. A. Jensen organiserer sit arbejde på den måde at store opgaver (defineret ved at tage længere tid end 10 minutter) sættes i en kø, hvorimod små opgaver (10 minutter eller derunder) lægges på en stak. Opgaverne ekspederes i følgende rækkefølge: Næste opgave tages altid fra stakken, hvis der er nogen, ellers tages den fra køen.

Du skal her konstruere et program som simulerer en typisk arbejdsdag for A. Jensen, og opgaven er delt op i en række delspørgsmål, hvor det første hjælper med til at forstå problemstillingen, og de øvrige leder frem til en struktureret 1øsning.

Spørgsmål 2.1
Konstruér en historie med et antal indkommende opgaver (hvor store de er og på hvilke tidspunkter), og lav en håndsimulering af A. Jensens arbejdsgang ved at tegne køen og stakken på en tavle eller et stykke papir. Prøv at variere historien så I får et indblik i, i hvilke situationer nogle opgaver kommer til at vente urimeligt længe.

Spørgsmål 2.2
Skriv en klasse af objekter som repræsenterer kopieringsopgaver, hvor vi af nemhedshensyn kun noterer information om hvor lang tid den enkelt opgave vil tage. Udstyr klassen med konstruktor, toString()-metode og to boolske metoder isSmall() og isBig() som klassificerer opgaven som beskrevet ovenfor.

Spørgsmål 2.3
Skriv en klasse som kan bruges til at opbevare de kopieringsopgaver som A. Jensen på en givet tidspunkt har liggende og venter på at blive ekspereret.
Udstyr den med metoder

- insert(<kopieringsopgave>) som indsætter en ny opgave
- nextJob() som udtager den næste opgave som skal ekspederes (NB: Det er praktisk at lade den returnere null hvis der ike er nogen opgaver).
- print(), som skriver den aktuelle samlig kopieringsopgaver ud (så man kan se, hvad der er i kø og på stak).

Det foreslås, at man benytter java.utils klasser LinkedList og Stack.

Spørgsmål 2.3
Vi skal nu skrive et program som simulerer en typisk arbejdsdag for A. Jensen; han arbejder ca. 8 timer, og da han er på slankekur springer han frokosten (og andre pauser) over. Vi antager et ur, som tæller hele minutter og som starter ved kl. 0. Simuleringen foretages efter følgende abstrakte algoritme (som hverken benytter en yderligere tids-kø af »begivenheder« eller en eksplicit klokke-variabel). Bemærk at de »ca. 8 timer« skal forstås sådan, at A. Jensen ikke modtager ny opgaver efter 8 timer, men han gør alle opgaver færdige inden han går hjem.

Det anbefales at du sætter dig grundigt ind i algoritmens principper, i stedet for blot at omskrive den til kritikløst Java (prøv at håndkøre den på de data du benyttede i spg. 2.1).

Variable:

næsteOpgaveAnkommer: *Det tidspunkt ude i fremtiden, hvor det er forudset, at næste kunde kommer og afleverer en ny opgave. (Hvis værdien er kommet op på 480 eller derover betyder det, at denne nye opgave ikke kommer indenfor døren.)*

næsteOpgaveKanPåbegyndes: *Det tidspunkt ude i fremtiden, hvor A.Jensen kan påbegynde en ny opgave. (Men der skal selvfølgelig være en opgave, for at han kan gå løs på den.)*

Algoritme:

```
næsteOpgaveKanPåbegyndes = 0;
næsteOpgaveAnkommer = 0; // Dvs. første kunde antages at komme kl. 0

Gentag følgende:
        Hvis    næsteOpgaveAnkommer ≤ næsteOpgaveKanPåbegyndes
                 og næsteOpgaveAnkommer < 480, så
          {  Generér en ny opgave med tilfældig varighed mellem 1 og 30 minutter;
             Sæt opgave i kø eller stak som beskrevet;
             Generér et tidsinterval med tilfældig varighed D mellem 2 og 35 minutter
                        for hvor længe der er til næste opgave ankommer;
             Sæt næsteOpgaveAnkommer = næsteOpgaveAnkommer + D;
          }
        Ellers // dvs.  næsteOpgaveKanPåbegyndes < næsteOpgaveAnkommer
            Hvis der er en opgave at vælge,
                 Tag en opgave ud af kø eller stak som beskrevet, og kald dens varighed V;
                 Sæt næsteOpgaveKanPåbegyndes = næsteOpgaveKanPåbegyndes + V;
            Ellers // Der er ingen opgave som venter
                 Hvis næsteOpgaveAnkommer ≥ 480
                          // Fyraften og alle opgaver, som skal løses, er løst
                          Stop
                 Ellers
                          // Der ingen opgaver nu, men vi ved der ankommer en om lidt
                          næsteOpgaveKanPåbegyndes = næsteOpgaveAnkommer
```

Sæt testudskrifter på, så man ved hver gennemløb af løkken kan se mængden af ventede opgaver og værdierne. Du skal vedlægge det output, dit program har producere; det er dog OK at sakse i det så kun starten og slutningen kommer med.