Universitetet i Agder
Fakultet for økonomi- og samfunnsfag

E K S A M E N

Emnekode:	**IS-207**
Emnenavn:	**Algoritmer og datastrukturer**
Dato:	16. mai 2008
Varighet:	0900-1300
Antall sider inkl. forside:	7
Målform:	Norsk
Tillatte hjelpemidler:	Alle trykte og skrevne
Merknader:	

Innledning

I denne oppgaven skal vi starte med å se på et enkelt kontrollprogram for en heis. Heisen har et panel hvor passasjerene i heisen kan velge etasjen de skal til, og det er en knapp ved heisdøra i hver etasje som brukes til å få heisen til å komme dit.

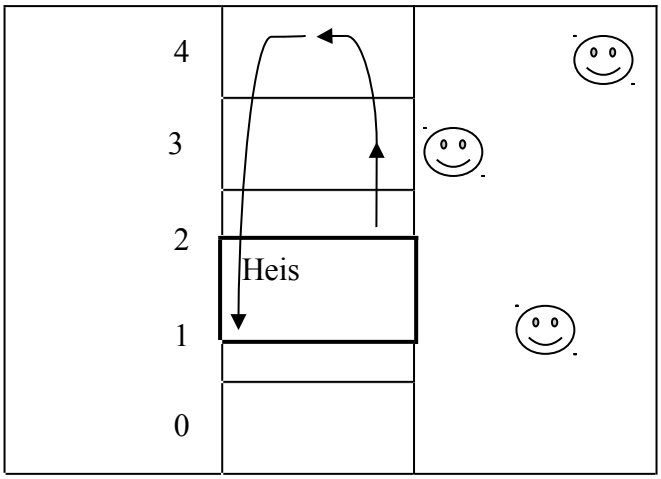
Programmet som kontrollerer heisen må balansere to viktige hensyn:

- Heisen må utnyttes best mulig. Dvs. flytte flest mulig passasjerer og bevege seg minst mulig uten passasjerer.
- Men dette skal helst skje uten at ventetiden for noen enkeltpassasjer blir for lang.

Det finnes en enkel algoritme, som er kjent som Elevator-algoritmen, som fortsatt brukes til å kontrollere noen heiser. Den kan formuleres slik:

Heisen beveger seg i samme retning så lenge noen skal av eller på i den retningen. Deretter snur den og tar dem som skal av/på i den andre retningen. Husk at noen kan trykke på knappene mens heisen er i bevegelse.

Et eksempel: Heisen er i første. Noen har trykket på knappen i tredje. Når heisen passerer andre trykker noen på knappen i første, deretter noen på knappen i fjerde. Heisen vil stoppe i tredje, fortsette til fjerde hvor den snur og går nedover igjen mot første. Hvis noen trykker på knappene 1-3 før heisen kommer dit vil den stoppe der på vei ned igjen.



Oppgave 1

Du skal implementere Elevator-algoritmen i java. Se klassen Elevator i vedlegg 1. Du kan anta at klassen inngår i et større program, som sjekker om noen har trykket på knapper, og bruker et Elevator til å kontrollere heisen.

a) **Valg av datastruktur (teller x %)**
Elevator trenger en intern datastruktur for å vite hvilke etasjer heisen skal stoppe i. Hva slags datastruktur vil du bruke? Begrunn svaret.

b) **Deklarasjoner (teller x %)**
Legg til deklarasjoner for datastrukturen i Elevator, og initialisering i constructoren.

c) **Registrere stoppønsker (teller x %)**
Hovedprogrammet kaller metoden nyttStopp() hver gang noen trykker på en knapp for å få heisen til å stoppe i en bestemt etasje (vi skiller ikke mellom bruk av knappene i heisen og knappene ved dørene). Metoden registrere at heisen skal stoppe i denne etasjen i datastrukturen fra 1b. Skriv ferdig metoden.

d) **Bevege heisen (teller x %)**
Hovedprogrammet kaller metoden flytt() for å få heisen til å flytte seg en etasje. (Heisen flyttes bare en etasje av gangen for å kunne ta hensyn til at noen trykker knapper mens heisen er i beveglese.) Bevegelsen skal følge elevator-algoritmen, dvs. heisen skal f.eks. fortsette opp så lenge det er registrert stopp høyere oppe. Hvis heisen har nådd en etasje hvor det er registrert stopp skal stoppet fjernes fra datastrukturen, fordi passasjerene har fått ønsket oppfylt. (Vi ser helt bort fra at dørene må åpnes og lukkes).
Hvis det ikke er flere stoppønsker igjen kan du velge om heisen skal fortsette (husk at den må snu i øverste og nederste etasje), eller om den skal gå til nederste etasje og stoppe der (husk at det i tilfelle kan være nødvendig å sette den i gang igjen når det kommer nye ønsker).

Oppgave 2 (teller x %)

Elevator-algoritmen brukes også til å kontrollere harddisker (den kalles også SCAN eller LOOK). . Disken er delt i sylindere, som tilsvarer etasjene for en heis, og all lesing (og skriving) utføres av et lesehode som tilsvarer heisen. Behovet for en slik algoritme kommer av at alle moderne operativsystemer tillater flere programmer (prosesser) å kjøre samtidig, så det kan være en kø av ønsker om å lese fra disken. Merk at i motsetning til heisen som ikke trenger å vite hvor mange som skal av i en etasje, må denne køen kunne inneholde flere ønsker om å lese (forskjellige filer) fra samme sylinder.

To alternative algoritmer er "nærmeste-først", og N-step-SCAN. Nærmeste-først flytter lesehodet til den nærmeste sylinderen noen ønsker å lese fra (uansett retning) og leser den ønskede filen,
N-step-SCAN har en kø av ønsker, tar ut de N første, og kjører elevator-algoritmen på dem. Nye ønsker som kommer inn blir lagt sist i køen og forstyrrer ikke elevator-algoritmen som bare holder på med de N første.

Sammenlign disse tre algoritmene med hensyn på de to kriteriene fra innledningen (som kan omformuleres slik når vi holder på med disker):

- lesehodet skal bevege seg minst mulig mellom hver fil som leses
- ventetiden for den prosessen som må vente lengst skal være kortest mulig

Hint: To "interessante" scenarioer er: Leseønske i den ene enden av disken, mens mange kommer inn i den andre, og nytt leseønske kommer for sylinder lesehodet nettopp har beveget seg forbi.

Oppgave 3

Klassen DiskController (vedlegg 2) er en halvferdig simulering av en disk-controller som bruker N-step-Scan algoritmen. Du skal skrive ferdig metoden scan() som lese N filer fra disken med minst mulig flytting av lesehodet.

a) Leseønskene i parameteren buf ligger i tilfeldig rekkefølge. Beskriv hva slags datastruktur og/eller algoritme(r) du vil bruke for å ordne dem i optimal rekkefølge for lesing.
   Hint: For å bruke elevatoralgoritmen må ønskene ordnes i stigende "avstand" fra lesehodet. "Avstanden" kan være til enden og tilbake igjen for noen ønsker (se figuren i innledningen).

b) Skriv ferdig metoden scan(). (Det er gjort noen forenklinger: Leseønsker er representert bare med sylindernummer, mens de i virkeligheten ville inneholde informasjon om hvor mye som skal leses, og nøyaktig hvor innenfor sylinderen.) Scan() skal kalle move() for å flytte lesehodet til en annen sylinder, og read() for å simulere "lesing" av en fil

Vedlegg 1 – class Elevator

```
/**
 * Write a description of class Elevator here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Elevator
{
    // Konstanter for retning
    public final static int NED = -1;
    public final static int STOP = 0;
    public static final int OPP = 1;

    /** Totalt antall etasjer, nederste etasje er 0,
     * øverste er antEtasjer - 1 */
    int antEtasjer;

    /** Etasjen heisen er i nå */
    int iEtasje;

    /** Retningen heisen beveger seg i */
    int retning;

    /** Datastruktur oppgave 1b */

    public Elevator(int antEtasjer) {
        this.antEtasjer = antEtasjer;
        iEtasje = 0;
        retning = OPP;

        // opprette datastruktur oppgave 1b
    }

    /** Registrer at heisen må stoppe fordi noen vil på eller av
     * @param etasje - etasjen heisen skal stoppe i */
    public void nyttStopp(int etasje) {
        // oppgave 1c
    }

    /** Flytter heisen en etasje opp eller ned. Heisen fortsetter i
     * samme retning så lenge det er ønsker om stopp i den retningen.
     * Heisen snur (begynner å gå den andre veien)når det ikke er flere
     * ønsker igjen i bevegelsesretningen.
     * @return etasjen heisen har flyttet til */
    public int flytt() {
        // oppgave 1d
        return iEtasje;
    }
}
```

Vedlegg 2 – class DiskController

```
import java.util.LinkedList;


/** Enkel simulering av en diskcontroller som bruker n-step-scan */
public class DiskController {
    // Køen av leseønsker
    // Nye ønsker som kommer inn legges på magisk vis sist i denne køen
    private LinkedList<Integer> queue;

    // Antall ønsker som behandles av gangen
    private final int N;

    // Sektoren hvor lesehodet er nå
    private int currentCylinder;

    /** Konstruktøren setter opp objektet */
    public DiskController(int n) {
        N = n;
        queue = new LinkedList<Integer>();
    }

    /** Legg til et ønske om å lese fra en bestemt sylinder
     * Kalles på magisk vis fra prosessene som vil lese filer */
    public void newRequest(int cylinder) {
        queue.add(cylinder);
    }

    /** Kalles for å start controlleren */
    public void run() {
        while (true) {
            // Hent N ønsker fra køen
            int[] buf = new int[N];
            for (int i=0; !queue.isEmpty() && i<N; i++) {
                buf[i] = queue.remove();
            }
            scan(buf);
        }
    }

    /** Behandler leseønskene ved å kalle read() for hvert ønske.
     * Flytter lesehodet minst mulig mellom hvert kall på read().
     * @param buf - array med cylindenr det skal leses fra */
    private void scan(int[] buf) {
        // oppgave 3
    }

    /** Simulerer flytting av lesehodet.
     * @param cylinder - sylindernummeret.
     */
    private void move(int cylinder) {
        if (cylinder != currentCylinder) {
            System.out.println("Flytter lesehodet til sylinder "+cylinder);
            currentCylinder = cylinder;
        }
    }

    /** Simulerer lesing av en fil */
    private void read() {
```

```
            System.out.println("Leser fil fra sylinder "+currentCylinder);
        }
    }
```