UNIVERSITETET I AGDER

FAKULTET FOR ØKONOMI OG SAMFUNNSVITENSKAP

E K S A M E N

Emnekode: **IS-207**
Emnenavn: **Algoritmer og datastrukturer**

Dato:
Varighet: 4 timer

Antall sider inkl. forside:

Tillatte hjelpemidler: Alle trykte og skrevne

Merknader:

Innledning

I det siste har det blitt skrevet en del i avisene om at passasjerer må stå igjen på holdeplassene fordi det ikke er nok plass på bussen. Denne oppgaven tar for seg hvodan man kan bruke event-simulering til å planlegge buss-kapasiteten bedre.

I event-simulering lager man modeller av problemdomenet hvor alle endringer i modellens tilstand skjer momentant, som en konsekvens av en hendelse (event). Eksempler på slike hendelser er en student som melder seg på et kurs, eller en passasjer som går på en buss. Studenten tar opp en plass på kurset, passasjeren tar opp en plass på bussen. Hendelsen kan føre til andre hendelser i framtida. Studenten kan f.eks. levere oppgaver, trekke seg fra kurset osv. Passasjerene på bussen (som vi skal konsentrere oss om heretter) kommer til å gå av bussen når de er kommet dit de skal.

Oppgave 1

Vi skal simulere transporten langs en bussrute i en retning. Det går busser med faste intervaller, f.ek. hver time, fra den ene endestasjonen til den andre. For å forenkle litt antar vi at bussen stopper på alle holdeplassene, og at det er nøyakig samme avstand mellom alle holdeplassene. Vedlegg 1 inneholder en enkel event-simulering av kun bussene.

Klassen BusSim må holde oversikt over de hendelsene (representert ved klassen Event) som skal skje i framtida. BussSim tar imot og lagrer framtidige hendelser. Hendelsene behandles i stigende rekkefølge, basert på tidspunktet for hendelsen. Det som skal skje først skjer først, kort og godt.

a) Hva slags datastruktur vil du bruke til å lagre Event-objektene i BusSim? Begrunn svaret.
b) Skriv ferdig deklarasjoner og setup() metoden i klassen BusSim.
c) Skriv ferdig metodene addEvent() og nextEvent() i BusSim.
d) Skriv ferdig eventuelle andre endringer i BusSim og/eller andre klasser

1a) Det beste valget her er prioritetskø (de vil kanskje skrive Heap), med tidspunktet for hendelsen som nøkkel. Hovedsaken er at det er lett å finne den neste hendelsen (den med minste tid). Binære søketrær vil fungere, men bli skeve og trege etter hvert.

Hvis de velger prioritetskø vil b, c, og d bli noe slikt:

public class BusSim {
… utelatt det som ikke har med denne oppgaven å gjøre

```
//// Oppgave 1a datastruktur
private PriorityQueue<Event> eventQ;

private void setup() {
    eventQ = new PriorityQueue<>();

    // start noen busser – ikke nevnt I oppgaven, bonus for å tenke på det
    for (int i=0; i<NUM_BUSES; i++) {
        Bus b = new Bus(i);
        Event e = new Event(i*30, b, 0);
        addEvent(e);
    }

    // plasser uten noen passasjerer – do.
    for (int i=0; i<NUM_STOPS; i++) stop[i] = 15;
}

public void addEvent(Event e){
    eventQ.add(e);
}

public Event nextEvent() {
    return eventQ.poll();
}
```

For at dette skal fungere må Event implementere Comparable, med sammenligning på tid, så 1d) blir:

public **class Event** implements Comparable{

```
int t; // tidspunktet når hendelsen skal skje
public int compareTo(Event that) {
    return this.getTime() - that.getTime();
```

```
}
public int getTime() { return t; }
```

Oppgave 2

Simuleringen i oppgave 1 er veldig enkel og ikke særlig realistisk. Ingen passasjerer går av f.eks. For å få ut noe nyttig informasjon trenger vi en mer avansert modell. Vi innfører to nye klasser: Passenger representerer en passasjer. Passasjeren skal reise fra en holdeplass til en annen. BusStop represesenterer en holdeplass. Holdeplassen har en id, som også kan brukes som indeks i en array, som i vedlegg 1. Det gjør det lettere å finne nest stopp. På holdeplassen står ingen, en eller flere passasjerer og venter på bussen. Når bussen kommer til en boldeplass

- går passasjerene som er kommet dit de skal av bussen. Vi simulerer det ved å slette Passenger-objektene fra datastrukturen i bussen.
- Deretter tar bussen med så mange passasjerer fra holdeplassen som den har plass til. Vi simulerer dette ved å flytte Passenger-objekter fra holdeplassen til bussen.

I tillegg trenger vi noen subklasser av Event for å representere nye typer hendelser, f.eks. at en passasjer kommer til en holdeplass og begynner å vente på bussen. **Du trenger ikke skrive Event-klassene!**

a) Hva slags datastruktur vil du bruke til å lagre Passenger-objekter i BusStop. Begrunn svaret og oppgi hvilke forutsetninger du har gjort.
b) Hva slags datstruktur vil du bruke til å lagre Passenger-objekter i Bus. Begrunn svaret.
c) Skriv ferdig feltdeklarasjoner, constructorer og eventuelle hjelpemetoder i Bus og BusStop (det er tilstrekkelig å skrive de metodene du trenger i d)
d) Skriv ferdig metoden makeStop() i klassen Bus. Metoden skal simulere det som skjer når bussen stopper på en holdeplass (slik som det er beskrevet over). Du kan legge til flere metoder, i hvilken som helst klasse, hvis du trenger det.

Opg2a – Hvis vi antar at passasjerene er noenlunde siviliserte og står i kø, kan vi bruke en kø, basert på lenket liste (evt. Et ringbuffer, men linkedlist har ubegrenset kapasititet, som kan være et poeng hvis det er altfor få busser). Arraylist vil også fungere, men elementene i liste må flyttes hvis ikke bussen kan ta med alle.
Opg 2b - når bussen kommer til et stopp, må vi finne alle passasjerer som skal av der. To alternativer (minst): prioritetskø av passasjerer med holdeplass som nøkkel, eller en enkel liste av lister, hvor indeks i den første er holdeplassnr, og innholdet i den andre er de som skal av der. Prioritetskø krever at Bus impelementerer Comparable, eller at de bruker en Comparator
Opg 2c kommer på neste side

Du finner et skjelett av klassene i vedlegg 2. Noen hint

- Det finnes ett simulator-objekt, som klassevariabelen BusSim.instance peker på
- BusSim objektet har arrayer (eller arraylister som inneholder alle Bus og BusStop objektene
- Klassene Bus og BusStop har et felt, id, som inneholder indeksen til disse arrayene.

Oppgave 2 c og d

```
public class BusStop {
    LinkedList<Passenger> passQ;

    public BusStop() {
        passQ = new LinkedList<>();
    }

    public void add(Passenger p) {
        passQ.addLast(p);
    }

    public Passenger next() {
        return passQ.removeFirst();
    }

    public boolean isEmpty() {
        return passQ.isEmpty();
    }

    public int getNumWaiting() {
        return passQ.size();
    }

}

public class Bus {
    public static final int MAX_PASS = 40;
    private static int nextId;

    int id;
    ArrayList<Passenger>[] passengers;
    int numPass;

    public Bus() {
        id = nextId++;
        passengers = new ArrayList[BusSim.NUM_STOPS+1];
        for (int i=0; i<=BusSim.NUM_STOPS; i++) passengers[i] = new ArrayList<>();
        numPass = 0;
    }

    public void makeStopAt(int stopNum, int t) {
        // get rid of passengers who have got to their destination
        int leaving = passengers[stopNum].size();
        passengers[stopNum].clear();
        numPass -= leaving;

        // get new on board
        BusSim sim = BusSim.instance;
        BusStop busStop = sim.stop[stopNum];
        int boarded = 0;
        while (! busStop.isEmpty() && numPass<MAX_PASS) {
            add(busStop.next());
            boarded++;
        }
    }

    public void add(Passenger p) {
        passengers[p.toStop].add(p);
        numPass++;
    }
}
```

Vedlegg 1

```
public class BusSim {
// Sim parametere - disse kan endres på for å teste ut forskjellige scenarier
    public static final int NUM_STOPS = 10; // antall stoppesteder
    public static final int NUM_BUSES = 3; // antall busser
    public static final int MAX_PASS = 40;
    public static final int TIME_BETWEEN_STOPS = 10; // kjøretid mellom stops
    public static final int MAX_DELAY = 5; // max forsinkelse mellom to stop

// global peker til det eneste BusSim objektet
    public static final BusSim instance = new BusSim();
// antal passasjerer som venter på hver holdeplass
    int[] stop = new int[NUM_STOPS];
    public final Random random = new Random();

    //// Oppgave 1a datastruktur…

    private int t; // "Klokka"

    public static void main(String[] args) {
        instance.run();
    }

// Dette er motoren i simuleringen den behandler Event-ojektene i rekkefølge
// basert på tidspunkt. Når et Event blir utført kan det bli lagt til nye.
    public void run() {
        setup();
        while ( /* opg 1b er det flere ubehandle Events igjen */ ) {
            Event e = nextEvent();
            t = e.getTime(); // hopp fram I tid til hendelsen
            e.happen(); // og utfør den…
        }
    }

    private void setup() {
        /* opg 1b */

// grovt forenklede eksempeldata – lag noen busser og sett 15 passasjerer
// på hver holdeplass
        for (int i=0; i<NUM_BUSES; i++) {
            Bus b = new Bus(i);
            Event e = new Event(i*30, b, 0);
            addEvent(e);
        }
        for (int i=0; i<NUM_STOPS; i++) stop[i] = 15;
    }

// Legg til et event-objekt på vent til det skal behandles
    public void addEvent(Event e){ // opg 1c }

// ta ut neste Event I tidsrekkefølge
    Public Event nextEvent() { /* opg 1c
    }

// Beregn hvor lang tid en buss bruker til neste stopp
// Metoden legger en tilfeldig forsinkelse til kjøretiden for å simulere at noen
// busser er forsinket.
    public int timeToNextStop() {
        return TIME_BETWEEN_STOPS + random.nextInt(MAX_DELAY);
    }
}
```

```
/** Event representerer hendelser */
public class Event {
    int t; // tidspunktet når hendelsen skal skje
    Bus bus; // bussen som stopper
    int stopNo; //stoppested nr

    public Event(int t, Bus bus, int stop) {
        this.t = t;
        this.bus = bus;
        this.stopNo = stop;
    }

    public int getTime() {
        return t;
    }

// I denne simuleringen har vi bare en type Event, en buss som stopper på
// et stoppested. Det er lagt inn noen utskrifter her, så det er mulig å se
// konsekvensen av endringer, f.eks. sette opp flere busser, større busse osv.
    public void happen() {
        BusSim sim = BusSim.instance; // få tak i BusSim objektet
        if (stopNo < sim.NUM_STOPS-1) {

// ikke ved endestasjonen
// bussen kan ta med minimum av de som venter på holdepl og ledige seter
            int n = Math.min(sim.stop[stopNo], sim.MAX_PASS - bus.numPass);
            System.out.format("Buss %d tok opp %d pass fra holdepl. %d.\n",
                              bus.id, n, stopNo);
            bus.numPass = bus.numPass + n;
            sim.stop[stopNo] = sim.stop[stopNo] - n;
            if (sim.stop[stopNo] > 0)
                System.out.println(sim.stop[stopNo]+" pass. står igjen\n");
// når bussen kommer til neste stop
            int eta = t + sim.timeToNextStop();
// lag ny hendelse for neste stop
            Event e = new Event(eta, bus, stopNo + 1);
// og gi den til BusSim objektet.
            sim.addEvent(e);
        }

        else System.out.format("Buss %d kom til endestasj med %d pass\n",
                               bus.id, bus.numPass);
    }
}

// Minimal representasjon av en buss
public class Bus {
    int id;
    int numPass; // antal passasjerer på bussen

    public Bus(int id) {
        this.id = id;
        numPass = 0;
    }
```

Vedlegg 2

```
public class Bus {
// Maks antal passasjerer bussen kan ta
    public static final int MAX_PASS = 40;
    public int id;

    public Bus() {
//oppgave 2c
    }

/** Simuler et stop på holdeplass stopNum, på tidspunktet t */
    public void makeStop(int stopNum, int t) {
// henter simulator-objektet, og BustStop-objektet som representerer holdeplassen
        BusSim sim = BusSim2.instance;
        BusStop busStop = sim.stop[stopNum];

       // oppgave2d
    }


    // andre metoder – oppgave 2c/d
}

/** BusStop representerer en holdeplass */
public class BusStop {
    int id; // unik id – samme verdi som objektets indeks
            //  i Arrayen BusSim.instance.stop
// datastruktur – oppgave 2c

    public BusStop() {
//oppgave 2c
    }

// andre metoder oppgave 2c/d
}


public class Passenger {
// holdeplassnr (indeks), fra og til
    int fromStop;
    int toStop;
}
```