Dette er løsningsforslag til eksamensoppgavene. Det betyr at det fortsatt kan være feil i dem. Oppgave 3-5 er kopi av en ordentlig besvarelse. Vi synes den er så god, at vi legger den ut som et eksempel på en besvarelse som stod til A med glans. Likevel inneholder den sikkert feil.

Tusen takk til kandidaten som ga sin tillatelse til at vi publiserer den!

Oppgave 1

Typene er skrevet over variablene, mens navnene er skrevet under.

Her er en skisse til hvordan jeg tenkte man skulle gå fram, med en løsning til de 7 definisjonene man skulle lage. En type er enten en klasse, eller et interface. Alle de 7 typene Q, QA, QK, osv. skal da være en klasse- eller interface- definisjon.

Det stod i oppgaven at «objekter av alle klasser det kan lages objekter av, er med». Det var 7 typer og 4 objekter. Mao. 3 av typene må være grensesnitt eller abstrakte klasser.

Videre står det at tegninga er uttømmende, noe som betyr at objektet som fem typer peker til må være lengst nede i klassehierarkiet, mens de to objektene som kun tre typer kan peke til må være av klasser lengst oppe. Intet objekt kan pekes på av bare Q, så Q må enten være en abstrakt klasse eller et grensesnitt. Og Q må være aller øverst, siden variabler av type Q kan peke til alle objektene. Variabler av type QA kan peke tre objekter, noe som innebærer at QA også må være langt oppe i hierarkiet. Det samme må da også QK:

```
    a Q              ( forkortelse for abstract class Q{} )
QK          QA
```

Siden det ikke finnes noe "rent" QA-objekt eller G-objekt, må disse være abstrakte klasser eller grensesnitt. I tegninga over må da QA være abstrakt klasse, mens G må være et grensesnitt fordi G kan peke til både QK og QA. G kan ikke implementers av Q, fordi da måtte også et QAC-objekt kunne pekes på av G. Da får vi:

```
interf. G

       a Q
QK i G     a QA
```

Siden QK ikke kan peke på andre objekter, må det være slutten på den grenen. De resterende tre typer må da være subklasser til QA:

```
       a Q
QK i G     a QA
             QAB
             QAC
             QABC
```

Vi har alt funnet ut at QABC må være nederst. QAB og QAC kan ikke være slik at den ene er en subklasse av den andre. Dette fordi de to typene ikke kan peke på samme objekt. Da får vi

```
       a Q
```

QK i G a QA
QAC QAB

QABC må være subklasse til QAB, fordi en QAC peker ikke kan peke til QABC:

a Q
QK i G a QA
QAC QAB
QABC

G kan peke til QAB og QABC, men ikke til QAC. Det betry at også QAB må implementere G:

a Q
QK i G a QA
QAC QAB i G
QABC
interface G

Dette gir løsningen (skrevet i Java-kode):

```
// Oppgave 1 a

interface G {}

abstract class Q {}
class QK extends Q implements G {}
abstract class QA extends Q {}
class QAC extends QA {}
class QAB extends QA implements G {}
class QABC extends QAB {}

// slutt oppg 1a
```

I tillegg kan vi ha tre løsninger til, hvis vi lar Q vare et grensesnitt:
(til eksamen holdt det med én, selvfølgelig)

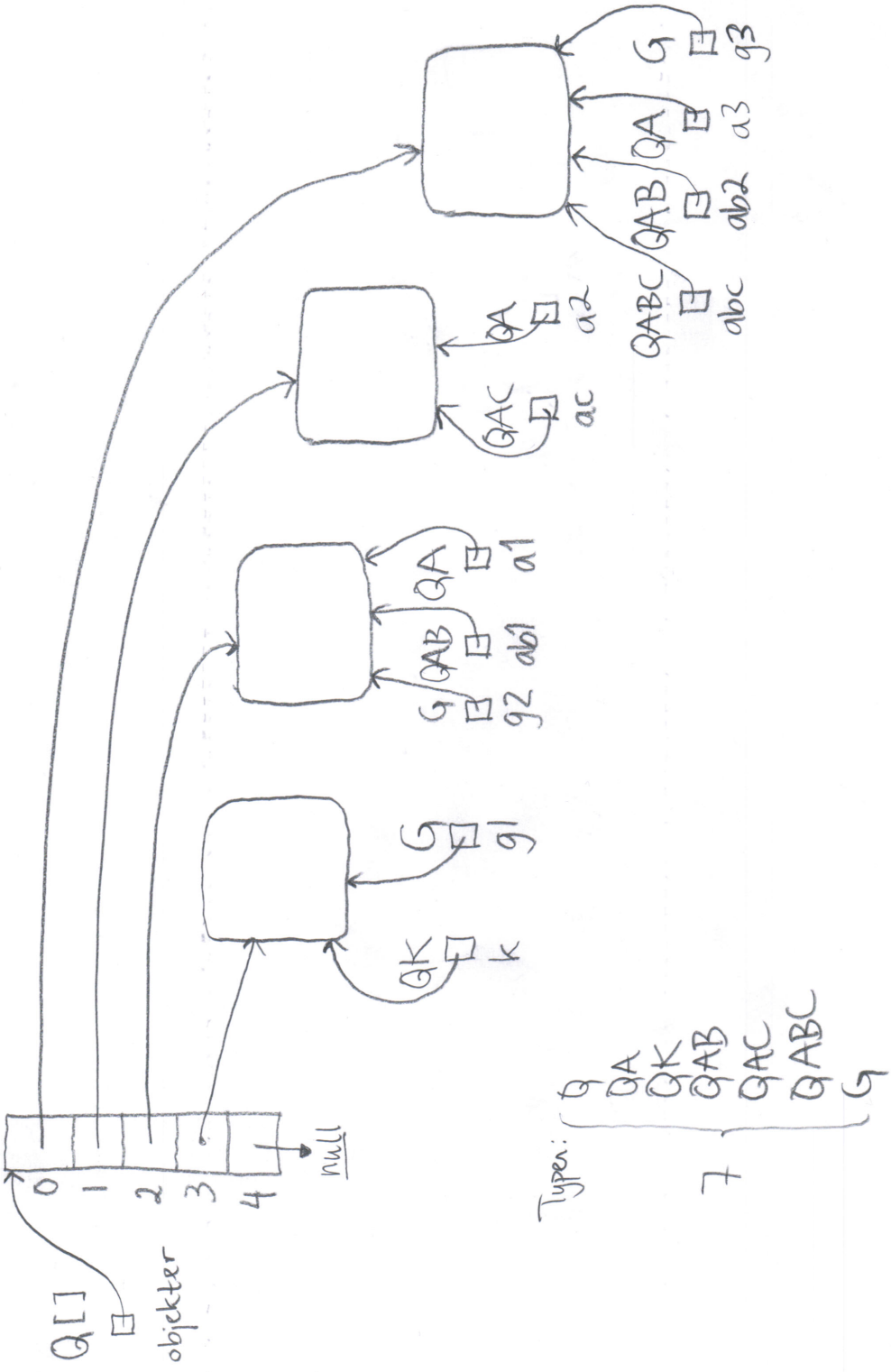
```
interface Q {}
interface G extends Q {}
interface QA extends Q {}
class QK implements G {}
class QAC implements QA {}
class QAB implements G, QA {}
class QABC extends QAB {}
```

Vi kan også erstatte en av G eller QA med en abstrakt klasse (men ikke begge):

```
interface Q {}
interface G extends Q {}
abstract class QA implements Q {}
class QK implements G {}
class QAC extends QA {}
```

```
class QAB extends QA implements G {}
class QABC extends QAB {}

// eller

interface Q {}
abstract class G implements Q {}
interface QA extends Q {}
class QK extends G {}
class QAC implements QA {}
class QAB extends G implements QA {}
class QABC extends QAB {}


// 1b, basert på det øverste forslaget til definisjoner

class Oppgave1 {
   public static void main(String[] args) {

      Q[] objekter = new Q[5];

      // objektet lengst til høyre
      //      objekter[0]:

      QABC abc =  new QABC();
      QAB ab2 = abc;
      QA a3 = abc; // eller a3 = ab2;
      G g3 = abc;
      objekter[0] = abc;

      // merk at rekkefølgen ikke er vilkårlig hvis
      // vi skal unngå å typekonvertere.

      //      objekter[1]:
      QAC ac = new QAC();
      QA a2 = ac;
      objekter[1] = ac;

      //      objekter[2]:

      QAB ab1 = new QAB();
      QA a1 = ab1;
      G g2 = ab1;
      objekter[2] = ab1;

      //      objekter[3]:

      QK k = new QK();
      G g1 = k;
      objekter[0] = k;
   }
}
```

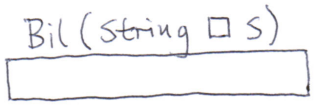
vedlegg 1


Q[]
objekter
0
1
2
3
4
null
G
g1
QK
k
G
QAB
QA
g2
ab1
a1
QAC
QA
ac
a2
QABC
QAB
QA
G
abc
ab2
a3
g3
Typer:
Q
QA
QK
QAB
QAC
QABC
G
7

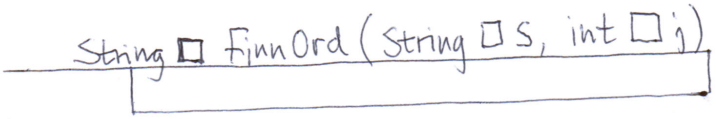
vedlegg 2

```
class TegneDS {
    public static void main(String[] args) {
        new ElemListe(args);
    }
}

class ElemListe {

    private Elem fElem = null;
    private Elem sElem = null;

    ElemListe ( String [] inputstrenger ) {
        Elem e = null;
        sElem = new Elem("Sisteelement?");
        fElem = sElem;
        for (String s: inputstrenger) {
            e = new Elem(s);
            sElem.neste = e;
            sElem = e;
        }
        for (String s: inputstrenger) {
            e = new Elem(s);
            e.neste = fElem;
            fElem = e;
        }

        /*
          OPPGAVE tegn datastrukturen slik den ser ut når
                programutførelsen har kommet hit
        */
    }

}

class Elem {
    static int ant = 0;

    Elem neste;
    String ord;
    int nr;

    Elem (String s) {
        nr = ant++;
        ord = s;
    }
}
```

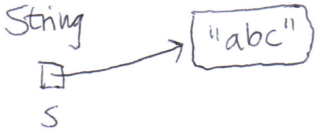
Konstruktøren i Bil:



En metode med to parametre og returverdi:

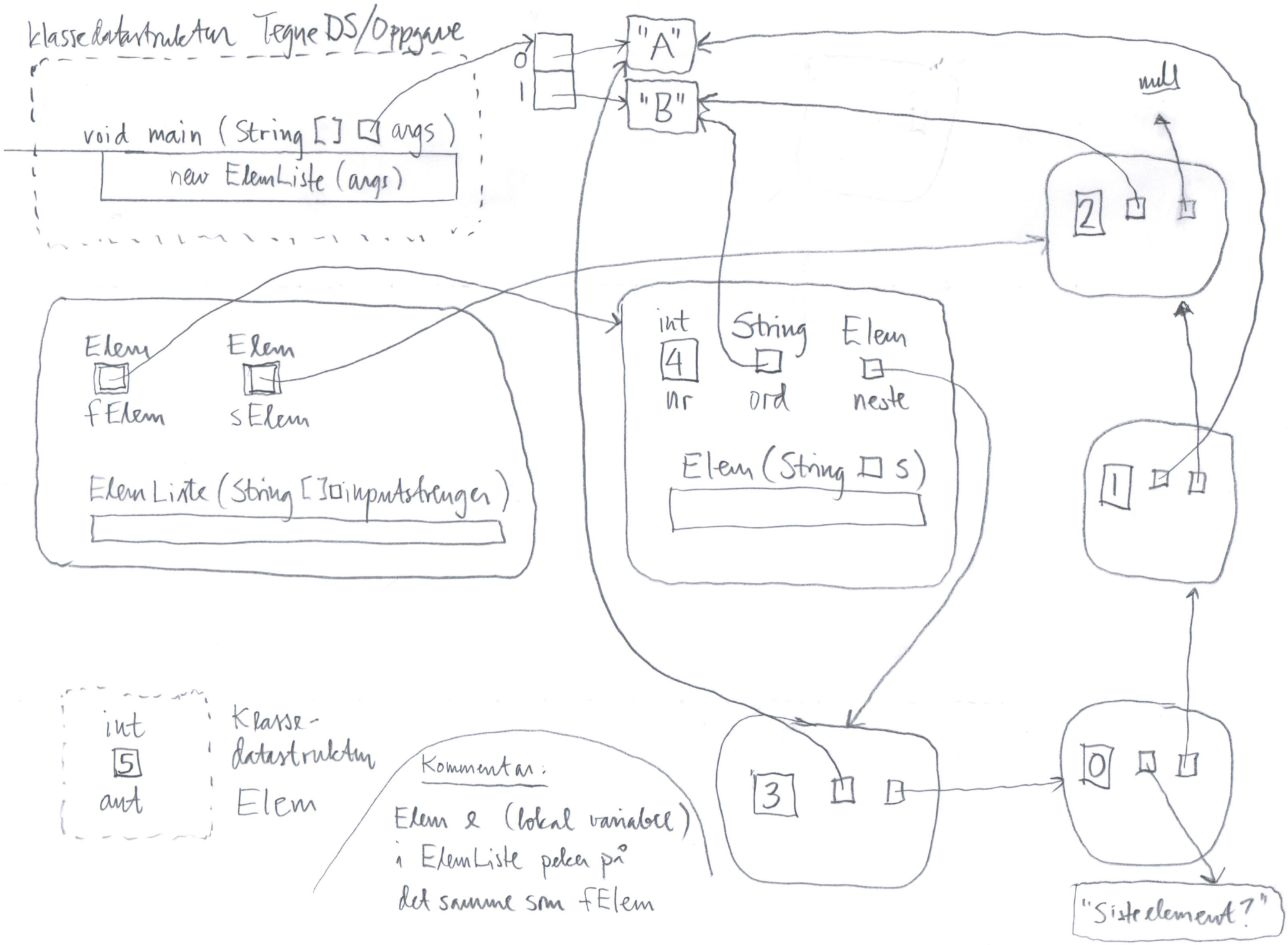


Et stringobjekt med verdien "abc":



Et objekt med en heltallsvariabel:

```
class Liste < T extends Comparable<T> > {

    private Node foran;

    private class ListeEnde extends Node {     }    // oppgave c

    private class Node {
        protected T t;   // peker til objektet som lagres i lista
        protected Node neste;

        Node (T nyttObjekt) {
            t = nyttObjekt;
        }

        int sammenlign(Node k) {  }                 // oppgave a

        void settInn(Node ny) {  }                  // oppgave b

        void skriv(){
            System.out.println(t);
            neste.skriv();
        }

    }

    Liste() {    }                                  // oppgave d



    public void settInn(T t) {      // du skal ikke endre denne
        Node nyNode = new Node(t);
        foran.settInn(nyNode);
    }

    public void skrivAlle() {
        System.out.println("Alle i lista:\n" + "---");
        foran.skriv();
        System.out.println(" --- SLUTT");
    }
}

class OrdnetLenkeliste {
    public static void main(String[] args) {
        Liste<String> ordliste  = new Liste<String>();

        String [] navn = new String []
            { "I", "dag", "er", "det", "eksamen", "i", "INF1010.\n",
              "Jeg", "håper", "du", "liker", "denne", "oppgaven.\n",
              "Lykke", "til!", "hilsen", "oppgaveforfatteren\n"};

        System.out.print("Setter inn: ");
        for (String n: navn) {
            System.out.print(n + " ");
            ordliste.settInn( n );
        }
        System.out.println();
        ordliste.skrivAlle(); System.out.println();
    }
}
```


klassedatastruktur TegneDS/Oppgave
void main (String [] args)
new ElemListe (args)
0
1
"A"
"B"
null
2
Elem
fElem
Elem
sElem
ElemListe (String [] inputstrengen)
int
4
nr
String
ord
Elem
neste
Elem (String s)
1
int
5
ant
Klasse-datastruktur
Elem
Kommentar:
Elem e (lokal variabel) i ElemListe peker på det samme som fElem
3
0
"Siste element?"

Skriv ikke her / *Do not write here*

③ a)
```
int sammenlikn(Node h){
    return t.compareTo(h.t);
}
```

// B og C kommer senere.

d)
```
public Liste(){
    ListeEnde hode = new ListeEnde(true);
    ListeEnde hale = new ListeEnde(false);
    foran = hode;
    foran.neste = hale;
}
```

Skriv ikke her / *Do not write here*

③ b)

```
void setInn(Node ny){
    if (neste != null){
        if (neste.sammenligne(ny) > 0){
            ny.neste = neste;
            neste = ny;
            return;
        }
        else{
            neste.setInn(ny);
        }
    }
}
```

c)

```
private class ListeEnde extends Node{
    boolean erNode;
    ListeEnde(boolean erNode){
        super(null);
        this.erNode = erNode;
    }
    int sammenligne(Node n){
        if (erNode){
            return -1;
        }else{
            return 1;
        }
    }
}
```

// fortsetter neste side

Skriv ikke her / *Do not write here*

③ b) # fortsettelse. Override skriv() -metoden.

```
void skriv(){
    if (!erHode){}
    else {
        super.skriv();
    }
}
} //slutt ListeEnde
```

```
// Merk1: siden vi aldri kaller hodet sin sammenlikn,
// kan Listeende godt kaun alltid returnere
// 1 i denne metoden, da den kalles kun halen
// Merk 2: testen (if neste != null) er egentlig
// unødvendig i settInn; da vi aldri vil nå
// enden fordi halen alltid vil være større.
```

Skriv ikke her / *Do not write here*

(4)

```
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
```

2)

```
public abstract class Skip {
    protected Brett brett;
    protected HRute startRute;
    protected int lengde;
    protected int antNede;
    public Skip(Brett b, HRute r, int lengde){
        brett = b;
        startRute = r;
        this.lengde = lengde;
        antNede = 0;
    }
}  //OBS: utvider m/ enkelstår metode, se neste oppg.
```

(4b)

```
public class VSkip extends Skip{
    public VSkip(Brett b, HRute r, int lengde){
        super(b, r, lengde);
        HRute[][] ruter = b.getRuteNett();
        for(int i=0; i<ruter.length; i++){
            for(int j=0; j<ruter[].length; j++){
                if(startRute == ruter[i][j]){
                    for(int h=0; h<lengde; h++){
                        ruter[i+h][j].settSkip(this);
                    }
                }
            }
        }
    }
} //slutt VSkip
```

Skriv ikke her / *Do not write here*

~~[illegible] Skip~~

```
public class HSkip extends Skip{
    public HSkip(Brett b, HRute r, int lengde){
        super(b,r,lengde);
        ~~[illegible]~~
        HRute[][] ruter = b.getRuteNett();
        for(int i=0; i<ruter.length; i++){
            for(int j=0; j<ruter.length; j++){
                if(sterkRute == ruter[i][j]){
                    for(int h=0; h<lengde; h++){
                        ruter[i][h+j].setHSkip(this);
                    }
                }
            }
        }
    }
}

// OBS: jeg utvider klassen Skip med følgende
  // metode

    public void enNede(){
        antNede++;
        if(antNede == lengde){
            brett.skipNede();
        }
    }
```

Skriv ikke her / *Do not write here*

(4) c)

```
public class HTrikk extends JLabel{
    Skip nittSkip;
    boolean tryhhet;

    public Skip giSkip(){
        return nittSkip;
    }
    public void settSkip(Skip s){
        nittSkip = s;
    }

    public void tryhh(){
        if (nittSkip != null){
            if (! tryhhet){
                nittSkip.enNede();
            }
        }
        tryhhet = true;
    }
}
```

Skriv ikke her / *Do not write here*

④ d)

```
public class Brett extends JFrame{
    HRute[][] ruter;
    ~~[illegible]~~
    ImageIcon skip;
    ImageIcon hav;
    int antSkip;
    int antSunkedeSkip;
    int antKlikk;
    ~~[illegible]~~
    MinLytter lytter;


    public Brett(){
        ruter = new HRute[10][10];
        skip = new ImageIcon("skip.png");
        hav = new ImageIcon("hav.png");
        antSkip = 7;
        antKlikk = 0;
        antSunkedeSkip = 0;
        lytter = new MinLytter();

        setLayout(new GridLayout(10,10));
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(300,300);

        // forts. neste ark
```

Skriv ikke her / *Do not write here*

```
// I Brett sin konstruktør:

for (int i = 0; i < 10; i++){
    for (int j = 0; j < 10; j++){
        HRute r = new HRute();
        r.setIcon(hav);
        r.addMouseListener(lyttHer);
        ruter[i][j] = r;
        add(r);
    }
}

HSkip h1 = new HSkip(this, ruter[0][0], 2);
HSkip h2 = new HSkip(this, ruter[2][1], 4);
HSkip h3 = new HSkip(this, ruter[8][5], 2);
HSkip h4 = new HSkip(this, ruter[9][1], 3);
VSkip v1 = new VSkip(this, ruter[4][3], 3);
VSkip v2 = new VSkip(this, ruter[0][7], 6);
VSkip v3 = new VSkip(this, ruter[5][9], 4);

setVisible(true);

} // slutt konstruktør
```

Skriv ikke her / *Do not write here*

```
private class MinLytter implements MouseListener{
    public void mouseClicked(MouseEvent e){
        HRute klikket = (HRute) e.getSource();
        if (klikket.getSkip() == null){
            antKlikk++;
        } else {
            klikket.setIcon(skip);
            antKlikk++; klikket.tryhh();
            if (antSkip == antSunkedeSkip){
                duHarVunnet();
            }
        }
    }
    public void mouseEntered(MouseEvent e){

    }
    // implementerer resten av metodene,
    // som tomme.
}
```

Skriv ikke her / *Do not write here*

```
// Resterende metoder i brett;

public void skipNede(){
        antSunkedeSkip++;
}

public Rute[][] getRuteVerdi(){
        return ruter;
}

public void duHarVunnet(){
        JOptionPane.showMessageDialog(null,
                "Du har vunnet!!!");
}

} // slutt Brett.
```

Skriv ikke her / *Do not write here*

```
import java.util.concurrent.*;

(5) a)  ~~public String[] sorterDel(String[]~~

        boolean
public ~~void~~ sorterDel(String[] a, int fom, int tom){
    boolean sortert = false; ~~boolean~~
    while (!sortert) {
        sortert = true;
        for (int i = fom; i < tom; i++){
            if (a[i].compareTo(a[i+1])>0){
                String hjelp = a[i];
                a[i] = a[i+1];
                a[i+1] = hjelp;
                sortert = false;
                gjordeNoe = true;
            }
        }
    }
    return gjordeNoe;
}
```

boolean gjordeNoe = false;

Kommer til nytte senere!

//OBS: Jeg antar i dette at metoden over er tilgjengelig i klassen Sortering.

Skriv ikke her / Do not write here

(5) d)

```
public class Sortering{
    String[] a;
    int antTraader;
    CountDownLatch nedteller;
    MinTraad[] traader;

    Sortering(){
        antTraader = 32;
        traader = new MinTraad[32];
    }


    public String[] sorter(String[] a){
        int lengde = a.length/antTraader;
        this.a = a;
        nedteller = new CountDownLatch(
            antTraader);
        for (int i=0; i<antTraader; i++){
            if (i==antTraader-1){
                MinTraad t = new MinTraad(
                    i*lengde, a.length-1);
                t.start()
            }
            else{
                MinTraad t = new MinTraad(
                    i*lengde,(i+1)*lengde);
                t.start();
            }
    }
```

Legger inn:
traader[i]=t;

Skriv ikke her / Do not write here

```
} //slutt for-løkke

try{
    nedteller.await();
}catch (InterruptedException e){
    e.printStackTrace();
    System.exit(1);
} //Avslutt alle tråder:
for (int i=0; i< antTraader; i++){
    traader[i].interrupt();
}
return a;
} //slutt metode
```

se trådklasse venter vah?

5c) //Lar hovedklassen (sortering) være monitoren.

```
public synchronized void byttIskjote(
int index1, int index2){
    String bytt = a[index1];
    a[index1] = a[index2];
    a[index2] = bytt;
    notifyAll();
    nedteller = new CountDownLatch(
        antTraader);
}
```

Skriv ikke her / *Do not write here*

```
private class MinTraad extends Thread{
    int fom;
    int tom;

    MinTraad(int fom, int tom){
        this.tom = tom;
        this.fom = fom;
    }

    public void run(){
        boolean sortert = false;
While(true) {   while (! sortert){
                sortert = true;
                if (a[fom].compareTo(a[fom+1]
                  >0){
                    sortert = false;
                    byttISkjote(fom, fom+1);
                }
                sortert += sorterDel(a, fom+1,
                    tom-1);
                if (a[tom-1].compareTo(a[tom
                    +1]) > 0) {
                    sortert = false;
                    byttISkjote(tom-1, tom);
                }

    }           // fortsetter neste sjekk
}
```

obs: legger inn for-løkke!

Vet ikke om noe ble endret!

Skriv ikke her / Do not write here

```
try {
    nedteller.countDown();
    wait();
} catch (InterruptedException e){}
    // nå er vi ferdig, og terminert
    teach();
}
} // slutt på while(true). Lønne
} // slutt
} // slutt trådklasse

} // slutt Sortering
```

5d: se prog over

Obs: oppgavene gjør over i hverandre.

^ del-

```
import java.util.Random;
import java.util.concurrent.CountDownLatch;

public class Oppgave5 {
	public static void main(String[] args) {
		new Oppgave();
	}
}

class Oppgave {

	private CountDownLatch barrier;

	Oppgave() {
		// Testdata
		Random r = new Random();
		char[] letters = "abcdefghijklmnopqrstuvwxyz".toCharArray();
		final int NUM = 64003;
		String[] values = new String[NUM];
		for (int i = 0; i < NUM; i++) {
			String word = "";
			for (int j = 0; j < 4; j++) {
				word += letters[r.nextInt(26)];
			}
			values[i] = word;
		}

		barrier = new CountDownLatch(1);

		// Oppgave b
		Monitor m = new Monitor(barrier);
		int start = 0;
		int per = values.length / 32;
		int stop = per;
		for (int i = 0; i < 32; i++) {
			new SortThread(values, start, stop, m).start();
			start = stop;
			stop = stop + per;
			if (i == 30) {
				stop = values.length - 1;
			}
		}

		try {
			barrier.await();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}

		for (String word : values) {
			System.out.println(word);
		}
	}
```

```
    // Oppgave b
    class SortThread extends Thread {
        private String[] values;
        private int start;
        private int stop;
        private Monitor m;

        public SortThread(String[] values, int start, int stopp, Monitor m) {
            this.values = values;
            this.start = start;
            this.stop = stopp;
            this.m = m;
        }

        public void run() {
            sort(values, start, stop, m);
        }
    }

    // Oppgave a
    public void sort(String[] a, int fom, int tom, Monitor m) {
        boolean changed;

        while (barrier.getCount() == 1) {
            for (int i = fom; i < tom; i++) {
                changed = false;
                for (int j = fom; j < tom; j++) {
                    if (a[j].compareTo(a[j + 1]) > 0) {
                        if (j == fom || j == tom - 1) {
                            m.change(a, j, j + 1);
                        } else {
                            String tmp = a[j];
                            a[j] = a[j + 1];
                            a[j + 1] = tmp;
                        }
                        changed = true;
                    }
                }
                if (!changed) {
                    m.waitForData();
                }
            }
        }
    }
}

class Monitor {
    private int counter = 0;
    private CountDownLatch barrier;

    Monitor(CountDownLatch barrier) {
        this.barrier = barrier;
    }
```

```
    public synchronized void waitForData() {
        try {
            counter++;
            if (counter == 32) {
                barrier.countDown();
            }
            wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    // Oppgave c
    public synchronized void change(String[] a, int i, int j) {
        String tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
        counter = 0;
        notifyAll();
    }

    public synchronized int getCounter() {
        return counter;
    }
}
```