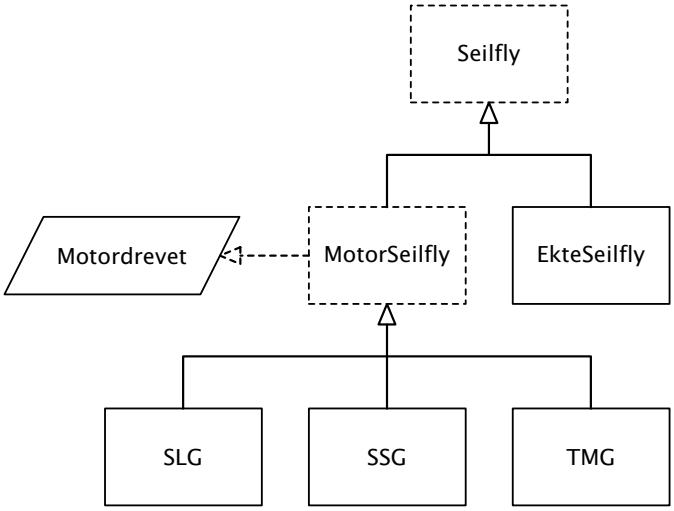
Konteeksamen i IN1010

Eric Jul* Dag Langmyhr†

16. august 2024 kl 9:00–13:00

Dette er et løsnings*forslag* så det finnes andre løsninger som er like gode og som vil gi full uttelling.

Oppgave 1



Oppgave 2–5

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/* Oppgave 2 */

abstract class Seilfly {
    final String id;
    final int glidetall, vingespenn;

    Seilfly neste = null, forrige = null;

    Seilfly (String defId, int defGlid, int defSpenn) {
        id = defId;  glidetall = defGlid;  vingespenn = defSpenn;
    }

    String hentId () {
        return id;
    }
```

*ericbj@ifi.uio.no
†dag@ifi.uio.no

```
    int hentGlidetall () {
        return glidetall;
    }

    int hentVingespenn () {
        return vingespenn;
    }

    /* Oppgave 4b */
    int finnStørsteVingespenn () {
        if (neste == null)
            return vingespenn;
        else
            return Math.max(vingespenn, neste.finnStørsteVingespenn());
    }
}


class EkteSeilfly extends Seilfly {
    EkteSeilfly (String defId, int defGlid, int defSpenn) {
        super(defId, defGlid, defSpenn);
    }
}


interface Motordrevet {
    int trekkraft ();
    String motortype ();
}


abstract class MotorSeilfly extends Seilfly implements Motordrevet {
    final int minTrekkraft;
    final String minMotortype;

    MotorSeilfly (String defId, int defGlid, int defSpenn,
                  int defKraft, String defMotor) {
        super(defId, defGlid, defSpenn);
        minTrekkraft = defKraft;  minMotortype = defMotor;
    }

    @Override
    public int trekkraft () {
        return minTrekkraft;
    }

    @Override
    public String motortype () {
        return minMotortype;
    }
}


class TMG extends MotorSeilfly {
    TMG (String defId, int defGlid, int defSpenn,
         int defKraft, String defMotor) {
        super(defId, defGlid, defSpenn, defKraft, defMotor);
    }
}
```

```
class SLG extends MotorSeilfly {
    SLG (String defId, int defGlid, int defSpenn,
         int defKraft, String defMotor) {
        super(defId, defGlid, defSpenn, defKraft, defMotor);
    }
}


class SSG extends MotorSeilfly {
    SSG (String defId, int defGlid, int defSpenn,
         int defKraft, String defMotor) {
        super(defId, defGlid, defSpenn, defKraft, defMotor);
    }
}


class Konkurransegruppe implements Iterable<Seilfly> {
    /* Oppgave 3a */
    Seilfly førsteFly, sisteFly;

    Konkurransegruppe () {
        // Egentlig unødvendig.
        førsteFly = null;  sisteFly = null;
    }

    /* Oppgave 3b */
    void leggTil (Seilfly f) {
        if (sisteFly == null) {
            // Formasjonen er tom.
            førsteFly = f;  sisteFly = f;
        } else {
            sisteFly.neste = f;  f.forrige = sisteFly;
            sisteFly = f;
        }
    }

    /* Oppgave 3c */
    boolean erMed (String id) {
        for (Seilfly fx: this)
            if (fx.hentId().equals(id)) return true;
        return false;
    }

    /* Oppgave 3d */
    Seilfly taUt (String id) {
        Seilfly detteFly = førsteFly;
        while (detteFly != null) {
            if (detteFly.hentId().equals(id)) {
                if (detteFly==førsteFly && detteFly==sisteFly) {
                    // Kun dette ene flyet i formasjonen.
                    førsteFly = null;  sisteFly = null;
                } else if (detteFly == førsteFly) {
                    // Det første flyet i formasjonen skal ut.
                    førsteFly = førsteFly.neste;  førsteFly.forrige = null;
                } else if (detteFly == sisteFly) {
                    // Det siste flyet i formasjonen skal ut.
                    sisteFly = sisteFly.forrige;  sisteFly.neste = null;
                } else {
```

```
                // Et fly inne i formasjonen skal ut.
                detteFly.forrige.neste = detteFly.neste;
                detteFly.neste.forrige = detteFly.forrige;
            }
            detteFly.forrige = null;  detteFly.neste = null;
            return detteFly;
        }
        detteFly = detteFly.neste;
    }
    return null;
}

/* Oppgave 3e */
@Override
public SeilflyIterator iterator () {
    return new SeilflyIterator();
}

class SeilflyIterator implements Iterator<Seilfly> {
    Seilfly detteSeilfly = førsteFly;

    @Override
    public boolean hasNext () {
        return detteSeilfly != null;
    }

    @Override
    public Seilfly next () {
        Seilfly res = detteSeilfly;
        detteSeilfly = detteSeilfly.neste;
        return res;
    }
}

/* Oppgave 3f */
Seilfly[] hentEkteSeilfly () {
    int antEkte = 0;
    for (Seilfly fx: this)
        if (fx instanceof EkteSeilfly) ++antEkte;

    Seilfly[] res = new Seilfly[antEkte];
    int n = 0;
    for (Seilfly fx: this)
        if (fx instanceof EkteSeilfly)
            res[n++] = fx;
    return res;
}

/* Oppgave 4a */
int besteGlidetall () {
    int best = 0;
    for (Seilfly fx: this)
        best = Math.max(fx.hentGlidetall(), best);
    return best;
}

/* Oppgave 4b */
int størstSpennvidde () {
    if (førsteFly != null)
        return førsteFly.finnStørsteVingespenn();
```

```
        else
            return 0;
    }

    /* Oppgave 4c */
    int[] histogramSpennvidde () {
        int[] res = new int[99+1];
        for (Seilfly fx: this) {
            int spenn = fx.hentVingespenn();
            if (spenn >= 10 && spenn <= 99)
                ++res[spenn];
        }
        return res;
    }
}


/* Oppgave 5a */
class KlarTilStart {
    ArrayList<Seilfly> klareSeilfly = new ArrayList<>();
    Lock laas = new ReentrantLock();
    Condition venterPåSeilfly = laas.newCondition();

    void seilflyklar (Seilfly f) {
        laas.lock();
        try {
            klareSeilfly.add(f);
            venterPåSeilfly.signalAll();
        } finally {
            laas.unlock();
        }
    }

    Seilfly nesteSeilfly () {
        laas.lock();
        try {
            while (klareSeilfly.size() == 0)
                venterPåSeilfly.await();
            return klareSeilfly.remove(0);
        } catch (InterruptedException e) {
            return null;
        } finally {
            laas.unlock();
        }
    }
}


/* Oppgave 5b */
class Plassjef implements Runnable {
    KlarTilStart monitor;
    Konkurransegruppe konkGruppe;
    int antSlepefly;

    Plassjef (int antFly, Konkurransegruppe gruppe, KlarTilStart mon) {
        antSlepefly = antFly;  konkGruppe = gruppe;  monitor = mon;
    }

    @Override
    public void run () {
```

```
        // Opprett slepeflyene:
        for (int i = 1;  i <= antSlepefly;  i++) {
            SlepePilot p = new SlepePilot(monitor);
            new Thread(p).start();
        }

        // Gjør alle seilflyene klare til sleping:
        for (Seilfly f: konkGruppe)
            monitor.seilflyklar(f);
    }
}


/* Oppgave 5c */
class SlepePilot implements Runnable {
    KlarTilStart monitor;

    SlepePilot (KlarTilStart mon) {
        monitor = mon;
    }

    @Override
    public void run () {
        while (true) {
            Seilfly f = monitor.nesteSeilfly();
            try {
                Thread.sleep(6 * 60 * 1000);  // Sov 6 minutter.
            } catch (InterruptedException e) { return; }
        }
    }
}
```