Løsningsforslag, gruppetime 11 (Tråder 1)

Oppgave 1 - Løsningsforslag (Kafé)

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

class Bord {
    private List<String> kopper = new ArrayList<>();
    private Lock lock = new ReentrantLock();
    private Condition ikkeTomt = lock.newCondition();
    private int totaltAntallKopper;

    public Bord(int totaltAntallKopper) {
        this.totaltAntallKopper = totaltAntallKopper;
    }

    public void serverKaffe(String kaffe) {
        lock.lock();
        try {
            kopper.add(kaffe);
            totaltAntallKopper--;
            ikkeTomt.signalAll();
        } finally {
            lock.unlock();
        }
    }

    public String hentKaffe() throws InterruptedException {
        lock.lock();
        try {
            while(kopper.isEmpty() && totaltAntallKopper > 0) {
                ikkeTomt.await();
            }
            if (!kopper.isEmpty()) {
                return kopper.remove(0);
            }
            return null;
        } finally {
            lock.unlock();
        }
    }
}
```

```
class Barista implements Runnable {
    private Bord bord; private int id, antKaffe;
    private final String[] drikker = {"Americano", "Café au lait", "Caffè latte", "Caffè
mocca", "Espresso", "Cortado"};

    public Barista(Bord bord, int id, int antKaffe) {
        this.bord = bord;
        this.id = id;
        this.antKaffe = antKaffe;
    }

    public void run() {
        Random rand = new Random();
        for (int i = 0; i < antKaffe; i++) {
            int indeks = rand.nextInt(drikker.length);
            String drikke = drikker[indeks];
            bord.serverKaffe(drikke);
            System.out.println("Barista " + id + " serverer en " + drikke);
        }
        System.out.println("Barista " + id + " er ferdig med å servere kaffe.");
    }
}

class Kaffedrikker implements Runnable {
    private Bord bord;
    private int id;
    public Kaffedrikker(Bord bord, int id) {
        this.bord = bord;
        this.id = id;
    }

    public void run() {
        int antallDrukket = 0;
        try {
            String kaffe;
            while ((kaffe = bord.hentKaffe()) != null) {
                System.out.println("Kaffedrikker " + id + " drikker en " + kaffe);
                antallDrukket++;
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            System.out.println("Kaffedrikker " + id + " ble avbrutt.");
        } finally {
            System.out.println("Kaffedrikker " + id + " har drukket " + antallDrukket +
" kopp(er) kaffe."); }
    }
}
```

```java
public class Hovedprogram {
    public static void main(String[] args) {
        final int antallBaristaer = 2;
        final int antallKopperPerBarista = 10;
        final int antallKaffedrikkere = 10;
        Bord bord = new Bord(antallBaristaer * antallKopperPerBarista);

        // Oppretter barista-traadene
        Thread[] baristaer = new Thread[antallBaristaer];
        for (int i = 0; i < antallBaristaer; i++) {
            baristaer[i] = new Thread(new Barista(bord, i + 1, antallKopperPerBarista));
        }

        // Oppretter kaffedrikker-traadene
        Thread[] kaffedrikkere = new Thread[antallKaffedrikkere];
        for (int i = 0; i < antallKaffedrikkere; i++) {
            kaffedrikkere[i] = new Thread(new Kaffedrikker(bord, i + 1));
        }

        // Starter barista-traadene
        for (Thread barista : baristaer) {
            barista.start();
        }

        // Starter kaffedrikker-traadene
        for (Thread kaffedrikker : kaffedrikkere) {
            kaffedrikker.start();
        }
    }
}
```

Oppgave 2 - Løsningsforslag (Brusautomat)

```java
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.Random;

class Brusautomat {
    private final int kapasitet;
    private int antBrusbokser;
    private int antBrusdrikkere;
    private final Lock laas = new ReentrantLock();
    private final Condition tom = laas.newCondition();
    private final Condition ikkeTom = laas.newCondition();
```

```java
public Brusautomat(int kapasitet, int antBrusdrikkere) {
    this.kapasitet = kapasitet;
    this.antBrusbokser = kapasitet;
    this.antBrusdrikkere = antBrusdrikkere;
}

public void refill() throws InterruptedException {
    laas.lock();
    try {
        while(antBrusbokser > 0) {
            if(antBrusdrikkere == 0) return;
            tom.await();
        }
        this.antBrusbokser = kapasitet;
        ikkeTom.signalAll();
        System.out.println("Brusautomaten er fylt opp!");
    } finally {
        laas.unlock();
    }
}

public void kjopBrus() throws InterruptedException {
    laas.lock();
    try {
        while(antBrusbokser == 0) ikkeTom.await();
        antBrusbokser--;
        System.out.println("En brus er kjopt!");
        if(antBrusbokser == 0) tom.signal();
    } finally {
        laas.unlock();
    }
}

public void ferdigAaDrikke() {
    laas.lock();
    try {
        antBrusdrikkere--;
        if(antBrusdrikkere == 0) tom.signalAll();
    } finally {
        laas.unlock();
    }
}

public boolean finnesFlereBrusdrikkere() {
    laas.lock();
    try {
        return antBrusdrikkere > 0;
```

```
        } finally {
            laas.unlock();
        }
    }
}

class Maskinfyller implements Runnable {
    private final Brusautomat brusautomat;

    public Maskinfyller(Brusautomat brusautomat) {
        this.brusautomat = brusautomat;
    }

    public void run() {
        try {
            while(brusautomat.finnesFlereBrusdrikkere()) {
                Thread.sleep(5000);
                brusautomat.refill();
            }
        } catch (InterruptedException e) {
            System.out.println("Maskinfylleren er avbrutt. Maskin stenges.");
            System.exit(1);
        }
    }
}

class Brusdrikker implements Runnable {
    private final Brusautomat brusautomat;
    private final int brusAaDrikke;
    public Brusdrikker(Brusautomat brusautomat, int brusAaDrikke) {
        this.brusautomat = brusautomat;
        this.brusAaDrikke = brusAaDrikke;
    }

    public void run() {
        try {
            for (int i = 0; i < brusAaDrikke; i++) {
                brusautomat.kjopBrus();
                Thread.sleep(new Random().nextInt(4) * 1000);
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            System.out.println("En brusdrikker ble avbrutt.");
        } finally { brusautomat.ferdigAaDrikke(); }
    }
}
```

```
public class Hovedprogram {
    public static void main(String[] args) {
        int kapasitet = 10;
        int antBrusdrikkere = 5;
        Brusautomat brusautomat = new Brusautomat(kapasitet, antBrusdrikkere);

        Thread[] brusdrikkere = new Thread[antBrusdrikkere];
        for (int i = 0; i < antBrusdrikkere; i++) {
            brusdrikkere[i] = new Thread(new Brusdrikker(brusautomat, new
Random().nextInt(11) + 5));
        }

        for (Thread brusdrikker : brusdrikkere) {
            brusdrikker.start();
        }

        Thread maskinfyller = new Thread(new Maskinfyller(brusautomat));
        maskinfyller.start();
    }
}
```

Oppgave 3 - Løsningsforslag (Sykehus)

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

class Sykehus {
    private final ArrayList<String> pasientkoe = new ArrayList<>();
    private final Lock laas = new ReentrantLock();
    private final Condition pasienterTilgjengelig = laas.newCondition();
    private int behandledePasienter = 0;
    private boolean erAlleLagtTil = false;

    void leggTilPasient(String pasientId) {
        laas.lock();
        try {
            pasientkoe.add(pasientId);
            System.out.println(pasientId + " lagt til i koen.");
            pasienterTilgjengelig.signalAll();
        } finally {
            laas.unlock();
        }
    }
```

```
    boolean behandlePasient() throws InterruptedException {
        laas.lock();
        try {
            while (pasientkoe.isEmpty() && !erAlleLagtTil) {
                System.out.println("Ingen pasienter i koe. Venter...");
                pasienterTilgjengelig.await();
            }
            if (pasientkoe.isEmpty() && erAlleLagtTil) {
                return false;
            }
            String pasientId = pasientkoe.remove(0);
            System.out.println("Behandler pasient " + pasientId);
            behandledePasienter++;
            return true;
        } finally {
            laas.unlock();
        }
    }

    void signaliserAllePasienterLagtTil() {
        laas.lock();
        try {
            erAlleLagtTil = true;
            pasienterTilgjengelig.signalAll();
        } finally {
            laas.unlock();
        }
    }
}

class MottaPasienter implements Runnable {
    private final Sykehus sykehus;
    private final int antallPasienter;

    MottaPasienter(Sykehus sykehus, int antallPasienter) {
        this.sykehus = sykehus;
        this.antallPasienter = antallPasienter;
    }

    @Override
    public void run() {
        for (int i = 1; i <= antallPasienter; i++) {
            sykehus.leggTilPasient("Pasient " + i);
        }
        sykehus.signaliserAllePasienterLagtTil();
    }
}
```

```java
class BehandlePasienter implements Runnable {
    private final Sykehus sykehus;

    BehandlePasienter(Sykehus sykehus) {
        this.sykehus = sykehus;
    }

    @Override
    public void run() {
        try {
            while (sykehus.behandlePasient()) {
                Thread.sleep(2000);
            }
        } catch (InterruptedException e) {
            System.err.println("Behandlingen ble avbrutt. Sykehuset maa stenge!");
            System.exit(1);
        }
    }
}
```