GRUPPE 1

Velkommen til
IN1010 – Uke 11

Plan for dagen

- Repetisjon tråder 1
- Barrierer
  - Join
  - CountDownLatch
- Conditions

Repetisjon – Låser

```java
class taUtTråd implements Runnable {
    Bank bank;

    public taUtTråd(Bank bank) {
        this.bank = bank;
    }

    @Override
    public void run() {
        bank.taUt(100);

    }
}
```

```java
public class Bank {
    int saldo = 100;

    public void taUt(int antall) {
        if (saldo >= antall) {
            saldo -= antall;
            System.out.println("Saldo: " + saldo);
        }
    }

    public int saldo() {
        return saldo;
    }

    Run | Debug
    public static void main(String[] args) {
        Bank bank = new Bank();

        Thread tråd1 = new Thread(new taUtTråd(bank));
        Thread tråd2 = new Thread(new taUtTråd(bank));

        tråd1.start();
        tråd2.start();
    }
}
```

```
Saldo: -100
Saldo: -100
```

```
Saldo: 0
```

Repetisjon – Låser

- For å forhindre slike feil, kan vi sørge for at kun en tråd har mulighet til å kalle på metoden/oppdatere en felles verdi av gangen

- Vi bruker objektet ReentrantLock, som implementerer Lock
  - Vi har metodene lock() og unlock()
  - Når låsen er låst, vil ingen andre tråder kunne aksessere den låste kodebiten
  - Andre tråder må vente til låsen låses opp

Repetisjon – Låser

```java
public class Bank {
    int saldo = 100;

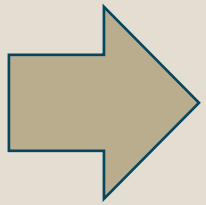
    public void taUt(int antall) {
        if (saldo >= antall) {
            saldo -= antall;
            System.out.println("Saldo: " + saldo);
        }
    }

    public int saldo() {
        return saldo;
    }

    Run | Debug
    public static void main(String[] args) {
        Bank bank = new Bank();

        Thread tråd1 = new Thread(new taUtTråd(bank));
        Thread tråd2 = new Thread(new taUtTråd(bank));

        tråd1.start();
        tråd2.start();
    }
}
```



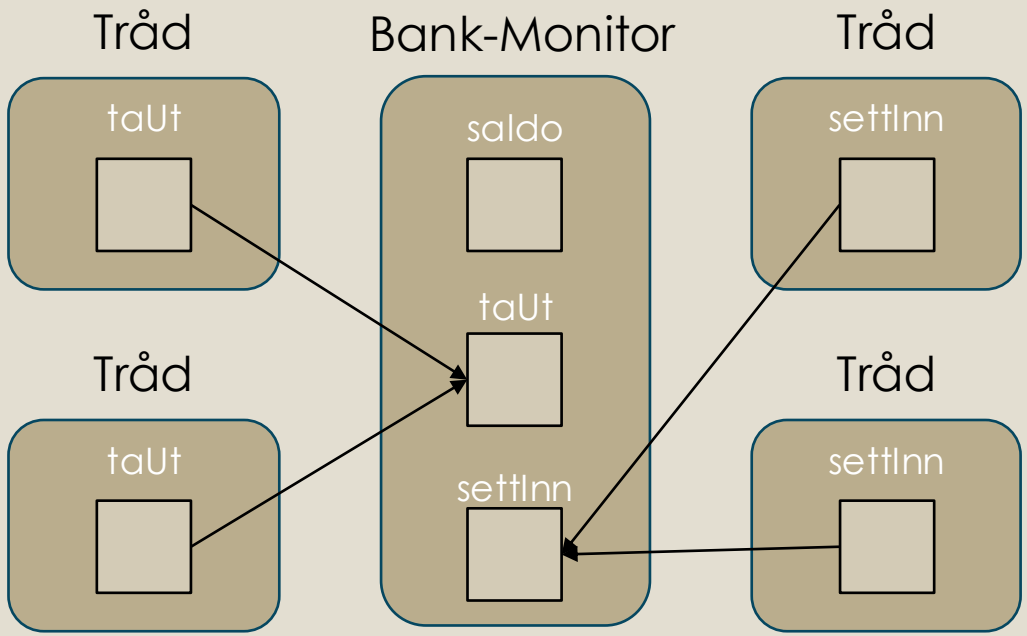
```java
public class Bank {
    int saldo = 100;
    Lock laas = new ReentrantLock();

    public void taUt(int antall) {
        laas.lock();
        try {
            if (saldo >= antall) {
                saldo -= antall;
                System.out.println("Saldo: " + saldo);
            }
        } finally {
            laas.unlock();
        }
    }

    public int saldo() {
        laas.lock();
        try {
            return saldo;
        } finally {
            laas.unlock();
        }
    }
```

Repetisjon – Monitor

- En klasse som håndterer kommunikasjon mellom tråder
- Tar kompleksiteten ut fra trådene
- Lagrer delt informasjon utenfor trådene



Huske regler for tråder

- Tråd metodene skal kun implementere logikk med bruk av monitor sine metoder som om monitor var et vanlig objekt

- Låsing, venting og all synkronisering skal håndteres i monitor klassen
- All delt data (variabler) skal ligge i monitor

- Tråder kan kalle:
  - sleep() – for å vente en viss tid
  - interrupt() – for å avslutte en annen tråd
  - join() – for å vente på at andre tråder skal bli ferdig

Barrierer

- En barriere lar oss sørge for at tråder kan møtes på et bestemt punkt i programmet
- Tar oss tilbake til sekvensiell kjøring når trådene er ferdig med arbeidet sitt
- Sørger for at ingen tråder går videre før de andre trådene er klare
- To typer i dette kurset
  - .join()
  - CountDownLatch

.join()

- En funksjon en tråd kan kalle på
- Gjør at tråden som kaller metoden, venter til tråden som skal joines er ferdig

```
public static void main(String[] args) {
    Bank bank = new Bank();

    Thread tråd1 = new Thread(new taUtTråd(bank));
    Thread tråd2 = new Thread(new taUtTråd(bank));

    tråd1.start();
    tråd1.join();

    tråd2.start();
    tråd2.join();

    System.out.println("Ferdig!");
}
```

CountDownLatch

- En CountDownLatch tar inn et heltall som parameter, som sier hvor mange ganger latchen må telles ned før barrieren kan passeres
- Alle tråder som kaller await() på latchen, må vente til latch har tellt ned nok ganger før den kan fortsette å kjøre
- Krever at vi vet hvor mange tråder vi vil vente på

```java
public static void main(String[] args) {
    CountDownLatch latch = new CountDownLatch(2);
    Bank bank = new Bank();

    Thread tråd1 = new Thread(new taUtTråd(bank, latch));
    Thread tråd2 = new Thread(new taUtTråd(bank, latch));

    tråd1.start();
    tråd2.start();
    try {
        latch.await();
    } catch (InterruptedException e) {

    }

    System.out.println("Ferdig!");
}
```

```java
class taUtTråd implements Runnable {
    Bank bank;
    CountDownLatch latch;

    public taUtTråd(Bank bank, CountDownLatch latch) {
        this.bank = bank;
        this.latch = latch;
    }

    @Override
    public void run() {
        bank.taUt(100);
        latch.countDown();
    }
}
```

Condition

- En condition hører til en lås
- Opprettes med lås.newCondition()
- En condition kan ventes på og signaliseres
  - condition.await() – vent på condition
  - condition.signal() – signaliser en tråd som venter på condition
  - condition.signalAll() – signaliser alle tråder som venter på condition

Condition

```java
class taUtTråd implements Runnable {
    Bank bank;

    public taUtTråd(Bank bank) {
        this.bank = bank;
    }

    @Override
    public void run() {
        try {
            bank.taUt(100);
        } catch (InterruptedException e) {
            System.err.println(e);
        }
    }
}
```

```java
public class Bank {
    int saldo = 100;
    Lock laas = new ReentrantLock();
    Condition nokSaldo = laas.newCondition();

    public void taUt(int antall) throws InterruptedException {
        laas.lock();
        try {
            while (saldo < antall) {
                nokSaldo.await();
            }
            saldo -= antall;
            System.out.println("Saldo: " + this.saldo());
        } finally {
            laas.unlock();
        }
    }

    public void settInn(int antall) {
        laas.lock();
        try {
            saldo += antall;
            nokSaldo.signalAll();
        } finally {
            laas.unlock();
        }
    }
```