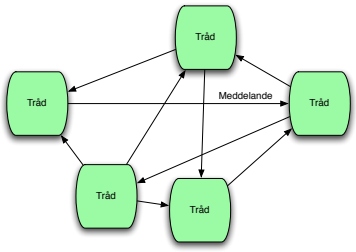
Trådar

Vad är trådar

- Tänk "flera program som exekverar samtidigt och tillsammans jobbar för att uppnå ett gemensamt mål"

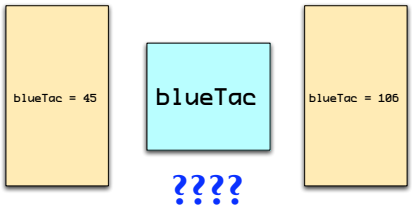


Problemet

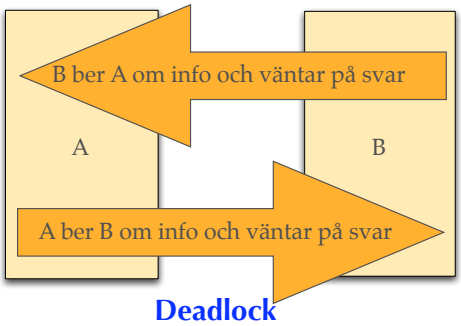
Två trådar tävlar om vem som får tillgång till en resurs

Problemet

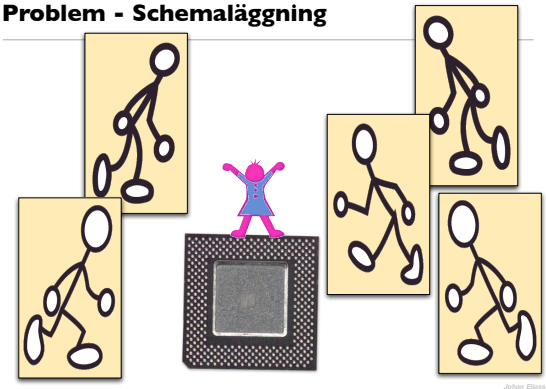
Två trådar tilldelar variabel värde. Vilket värde får den?

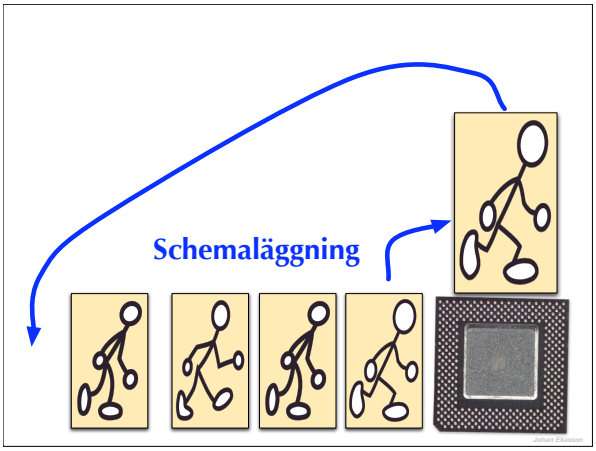


Problemet - Deadlock



Problem - Schemaläggning





Alltså ...

- Fördelarna är stora
- Problemen är signifikanta
- ... och det är kul!!!

De viktigaste "regelerna"

- Anta alltid att alla trådar/processer/program exekveras samtidigt
- Tro inte att du kan gissa i vilken ordning olika trådar/processer/program exekveras ... det funkar aldrig!!!
- Anta alltid att det värsta möjliga fallet uppstår

Java

- Använder sig av trådbegreppet
- Klassen - Thread
- Klassen - Runnable

```
public class MyThreadExample
    extends Thread
{
    private String name;

    public MyThreadExample( String name )
    {
        this.name = name;
    }

    private void doSomething( )
    {
        for( int i = 0 ; i < 100 ; i++ ){
            System.out.print( "Hi " );
            System.out.print( "there " );
            System.out.print( "world " );
            System.out.print( "from " );
            System.out.println( name );
        }
    }

    public void run()
    {
        doSomething();
    }
}
```

```
public class Demo
{
    public static void main( String[] args )
        throws InterruptedException
    {
        MyThreadExample kalle  = new MyThreadExample( "Kalle" );
        MyThreadExample knatte = new MyThreadExample( "Knatte" );
        MyThreadExample fnatte = new MyThreadExample( "Fnatte" );
        MyThreadExample tjatte = new MyThreadExample( "Tjatte" );

        kalle.start();
        knatte.start();
        fnatte.start();
        tjatte.start();

        kalle.join();
        knatte.join();
        fnatte.join();
        tjatte.join();
    }
}
```

```
Hi there world from Kalle
Tjatte
Hi Hi there world there world from Tjatte
Hi there world from Tjatte
from Kalle
Hi there world from Hi there world from Tjatte
Hi Kalle
there world from Hi there world from Kalle
Hi Tjatte
Hi there world there world from Kalle
from Tjatte
Hi there world Hi there world from Kalle
from Tjatte
```


Alternativ implementation

- Implementera gränssnittet Runnable


```
public class Counter
    implements Runnable
{
    private static int count = 0;

    public void run( )
    {
        for( int i = 0 ; i < 9 ; i++ ){
            count = i;
            System.out.println(
                "count = " + count +
                " i = " + i );
        }
    }
}
```



```
public class Demo
{
    public static void main ( String[] args )
        throws InterruptedException
    {
        Thread a = new Thread( new Counter( ) );
        Thread b = new Thread( new Counter( ) );
        Thread c = new Thread( new Counter( ) );
        Thread d = new Thread( new Counter( ) );
        Thread e = new Thread( new Counter( ) );

        a.start();
        b.start();
        c.start();
        d.start();
        e.start();

        a.join();
        b.join();
        c.join();
        d.join();
        e.join();
    }
}
```

```
count = 0 i = 0
count = 1 i = 1
count = 2 i = 2
count = 3 i = 3
count = 4 i = 4
count = 5 i = 5
count = 6 i = 6
count = 7 i = 7
count = 8 i = 8
count = 0 i = 0
count = 1 i = 1
count = 2 i = 2
count = 3 i = 3
count = 4 i = 4
count = 5 i = 5
```



```
java Demo
count = 0 i = 0
count = 1 i = 0
count = 1 i = 0
count = 1 i = 1
count = 2 i = 0
count = 1 i = 0
count = 1 i = 2
count = 3 i = 1
count = 2 i = 1
count = 2 i = 1
count = 2 i = 1
count = 2 i = 3
count = 4 i = 2
```

OOPS! Vad hände här???

Hur stoppar man en tråd

- Sätt en variabel som talar om att tråden ska sluta köra
- Metoden interrupt om tråden kan ligga och vänta på något

```
public class Demo
{
    public static void main( String[] args )
        throws InterruptedException
    {
        Simple kalle = new Simple( "Kalle" );
        Simple kajsa = new Simple( "Kajsa" );

        Thread t1 = new Thread( kalle );
        Thread t2 = new Thread( kajsa );

        t1.start();
        t2.start();

        Thread.currentThread().sleep(10);

        kalle.endThis();
        kajsa.endThis();
    }
}
```



```
public class Simple
    implements Runnable
{
    private boolean done = false;
    private String  name;

    public Simple( String name )
    {
        this.name = name;
    }

    public void run( )
    {
        while( ! done ){
            System.out.print( name + " " );
        }
        System.out.println( "\n\n\n" + name + " är klar");
    }

    public void endThis( )
    {
        done = true;
    }
}
```

Run och endThis körs i olika trådar!!

Timers

- TimerTask
- Timer
- Kan användas för att utföra händelser vid en senare tidpunkt eller upprepat med ett visst intervall
- Se exempel 5-8, 5-9

"Trådsäker"

- Lokala variabler är alltid säkra
- Instansvariabler beror på hur klassen används
- Klassvariabler är inte säkra

Synkronisering

- Ny modifierare: synchronized
- Bara en tråd kan köra i metoder som är syncronized i det aktuella objektet


```
public class MyThreadExample
    extends Thread
{
    private String name;

    public MyThreadExample( String name )
    {
        this.name = name;
    }

    private void doSomething( )
    {
        for( int i = 0 ; i < 100 ; i++ ){
            System.out.print( "Hi " );
            System.out.print( "there " );
            System.out.print( "world " );
            System.out.print( "from " );
            System.out.println( name );
        }
    }

    public void run()
    {
        doSomething();
    }
}
```

```
public class MyThreadExample
    extends Thread
{
    private String name;

    public MyThreadExample( String name )
    {
        this.name = name;
    }

    private void doSomething( )
    {
        for( int i = 0 ; i < 100 ; i++ ){
            print( name );
        }
    }

    private synchronized static void print( String name )
    {
        System.out.print( "Hi " );
        System.out.print( "there " );
        System.out.print( "world " );
        System.out.print( "from " );
        System.out.println( name );
    }

    public void run()
    {
        doSomething();
    }
}
```



```
Hi there world from Fnatte
Hi from Hi there there world from Fnatte
Hi there world from Fnatte
Hi there world Kalle
world from from Fnatte
Hi there world from Fnatte
Hi there world from Knatte
Hi Tjatte
Fnatte
Hi there Hi there there world from Fnatte
Hi there world from Fnatte
Hi there world from Fnatte
```

```
- java Demo
...
Hi there world from Kalle
Hi there world from Tjatte
Hi there world from Kalle
Hi there world from Tjatte
Hi there world from Kalle
Hi there world from Tjatte
Hi there world from Kalle
Hi there world from Tjatte
Hi there world from Kalle
Hi there world from Knatte
Hi there world from Fnatte
Hi there world from Tjatte
Hi there world from Fnatte
...
```

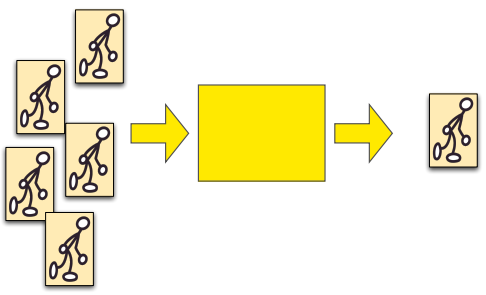

När?

- Effektivitet
- Inte allting
- Bara när det behövs

Synkronisera på objekt

```
private static Object lockObject = new Object();

private void doSomething( )
    throws InterruptedException
{
    for( int i = 0 ; i < 100 ; i++ ){
        synchronized ( lockObject ){
            System.out.print( "Hi " );
            sleep( r.nextInt(30) );
            System.out.print( "there " );
            sleep( r.nextInt(30) );
            System.out.print( "world " );
            sleep( r.nextInt(30) );
            System.out.print( "from " );
            sleep( r.nextInt(30) );
            System.out.println( name );
            sleep( r.nextInt(30) );
        }
    }
}
```


Enkel buffer




```
import java.util.Random;

public class Producer
  extends Thread
{
  private String name;
  private Buffer buffer;
  private Random rnd;

  public Producer( String name, Buffer buffer, long seed )
  {
    this.name = name;
    this.buffer = buffer;

    rnd = new Random( seed );
  }

  public void run( )
  {
    System.out.println( name + " startar" );
    try{
      for( int counter = 0 ; counter < 30 ; counter++ ){
        sleep( rnd.nextInt( 40 ) );
        buffer.leave( name );
      }
    } catch ( InterruptedException ie ){
      ie.printStackTrace();
    }
    System.out.println( name + " är klar" );
  }
}
```

```
import java.util.Random;
import java.lang.Thread;

public class Consumer
   implements Runnable
{
   Buffer buffer;

   public Consumer( Buffer buf )
   {
      buffer = buf;
   }

   public void run( )
   {
      int counter = 0;

      while( true ){
         System.out.println( counter++ + " " + buffer.fetch() );
      }
   }
}
```



```
import java.util.Random;

public class Demo
{
   private final static int NR_OF_PRODUCERS = 10;

   public static void main( String args[] )
   {
      Random rnd = new Random();
      Thread p[] = new Thread[NR_OF_PRODUCERS];
      Buffer b   = new Buffer();

      Thread c   = new Thread( new Consumer( b ) );
      c.start();

      for( int counter = 0; counter < NR_OF_PRODUCERS; counter++ ){
        p[counter] = new Thread( new Producer( "Person_" + counter, b, rnd.nextLong() ) );
        p[counter].start();
      }

   }
}
```



```
public class Buffer
{
   final private static int BUFFER_SIZE = 3;

   private String  data[]    = new String[BUFFER_SIZE];
   private int     firstfree = 0;
   private int     oldest    = 0;
   private boolean empty     = true;
   private boolean full      = false;

   private void addToBuffer( String name )
   {
      data[firstfree] = name;
      firstfree = (firstfree + 1) % BUFFER_SIZE;
      full = firstfree == oldest;
      empty = false;
   }

   private String getFromBuffer( )
   {
      String s = data[oldest];
      oldest = (oldest + 1) % BUFFER_SIZE;
      empty = firstfree == oldest;
      full = false;
      return s;
   }

   synchronized public void leave( String name )
   {
      boolean done = false;

      try{
         while( full ){
            System.out.println( name + " väntar" );
            wait();
         }
         addToBuffer( name );
         notifyAll();
      } catch ( InterruptedException ie ){
         ie.printStackTrace();
      }
   }

   synchronized public String fetch( )
   {
      String  s = "error";

      try{
         while( empty ){
            System.out.println( "Fetch väntar" );
            wait();
         }
         s = getFromBuffer();

         notifyAll();
      } catch ( InterruptedException ie ){
         ie.printStackTrace();
      }
      return s;
   }
}
```



```
final private static int BUFFER_SIZE = 3;

private String  data[]    = new String[BUFFER_SIZE];
private int     firstfree = 0;
private int     oldest    = 0;
private boolean empty     = true;
private boolean full      = false;
```



```
private void addToBuffer( String name )
{
   data[firstfree] = name;
   firstfree = (firstfree + 1) % BUFFER_SIZE;
   full = firstfree == oldest;
   empty = false;
}

private String getFromBuffer( )
{
   String s = data[oldest];
   oldest = (oldest + 1) % BUFFER_SIZE;
   empty = firstfree == oldest;
   full = false;
   return s;
}
```



```
synchronized public void leave( String name )
{
   boolean done = false;

   try{
      while( full ){
         System.out.println( name + " väntar" );
         wait();
      }
      addToBuffer( name );
      notifyAll();
   } catch ( InterruptedException ie ){
      ie.printStackTrace();
   }
}
```

```
synchronized public String fetch( )
{
   String  s = "error";

   try{
      while( empty ){
         System.out.println( "Fetch väntar" );
         wait();
      }
      s = getFromBuffer();

      notifyAll();
   } catch ( InterruptedException ie ){
      ie.printStackTrace();
   }
   return s;
}
```


Swing och trådar

- Swing kör i egen tråd
- Dina "action handlers" kan "blocka" GUIt
- Swing är inte trådsäkert


```
public class SwingTest
    extends JFrame
{
    JButton slow;
    JLabel  text;

    public SwingTest( )
    {

    }

    private void slowCounter( )
    {
        int counter = 0;

        try{
            for( int i = 0; i < 5 ; i++ ){
                counter++;
                text.setText( "Counter = " + counter );
                Thread.sleep( 1000 );
            }
        } catch( java.lang.InterruptedException e ) {
        }
    }
}
```


```
            setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
            slow = new JButton( "Slow" );
            JButton fast = new JButton( "Fast" );
            text = new JLabel();
            slow = new JButton( "Slow" );

            add( text, BorderLayout.NORTH );
            add( slow, BorderLayout.CENTER );
            add( fast, BorderLayout.SOUTH );

            text.setText( "Blipp" );

            fast.addActionListener( new ActionListener()
                {
                    public void actionPerformed( ActionEvent event )
                    {
                        text.setText( "Plopp" );
                    }
                }
            );

            slow.addActionListener( new ActionListener()
                {
                    public void actionPerformed( ActionEvent event )
                    {
                        slowCounter();
                    }
                }
            );

            pack();
            setVisible( true );
```



```
        private void slowCounter( )
        {
            Runnable task = new SwingWorker()
                {
                    private int counter = 0;

                    public void init( )
                    {
                        slow.setEnabled( false );
                    }
                    public void update( )
                    {
                        text.setText( "Counter = " + counter );
                    }
                    public void finish( )
                    {
                        text.setText( "Klart" );
                    }

                    public void work( )
                    {
                        try{
                            for( int i = 0; i < 30 ; i++ ){
                                counter++;
                                doUpdate();
                                Thread.sleep( 1000 );
                            }
                        } catch( java.lang.InterruptedException e ) {
                        }
                    }
                };
            Thread slowDancer = new Thread( task );
            slowDancer.start();
        }
```



```
abstract class SwingWorker
   implements Runnable
{
   public abstract void work() throws InterruptedException;

   public void init() {}
   public void update() {}
   public void finish() {}

public final void run()
{
   doInit();
   try
   {
      done = false;
      work();
   } catch (InterruptedException ex) {
   } finally {
      done = true;
      doFinish();
   }
}

private boolean done;
```

```
private void doInit()
{
   EventQueue.invokeLater(new
      Runnable()
      {
         public void run() { init(); }
      }
   );
}

protected final void doUpdate()
{
   if (done) return;
   EventQueue.invokeLater(new
      Runnable()
      {
         public void run() { update(); }
      }
   );
}

private void doFinish()
{
   EventQueue.invokeLater(new
      Runnable()
      {
         public void run() { finish(); }
      }
   );
}
```

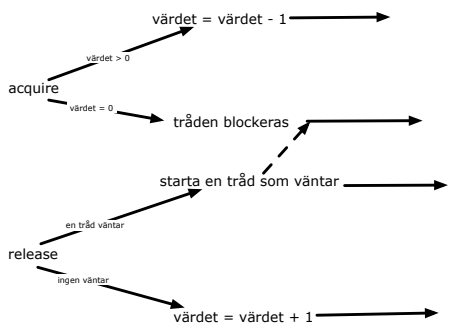
OBS!

En klass SwingWorker med samma syfte (men andra metodnamn) finns i java sedan 1.6

Använd den istället

Ett annat sätt att synkronisera

- Semaforer
  - Atomisk access
  - Semaforen s kan sättas till N
  - acquire
  - release



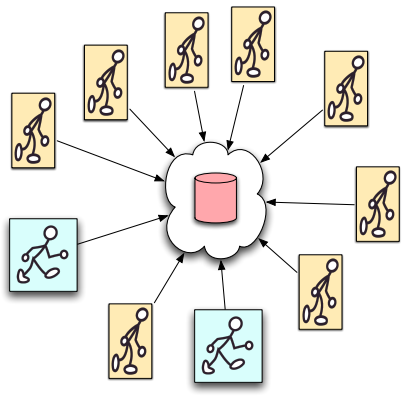

```
Semaphore controllerAccess   = new Semaphore(1);

controllerAccess.acquire();

controllerAccess.release();
```

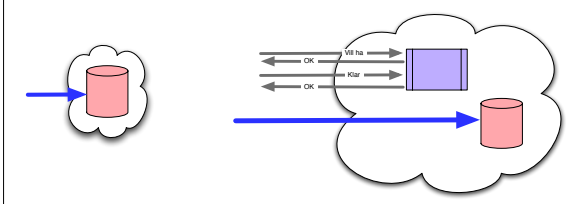

Problem för er

- Hur hindrar man att flera trådar ändrar på ett objekt samtidigt?

Problem

- En "databas" där man vill kunna tillåta ett godtyckligt antal trådar som läser.
- Bara en tråd åt gången får ändra på "databasen"






```
public class SomeCriticalRegion
{
   public void readStuff( )
   {

   }

   public void writeStuff( )
   {

   }
}
```



```
    for(;;){
      System.out.println( name + " wants permission to enter region" );
      control.readerEnterRegion();
      System.out.println( name + " got permission to enter region" );

      System.out.println( name + " enters region" );
      region.readStuff( );
      waitAMoment();
      System.out.println( name + " leaves region" );

      System.out.println( name + " wants to hand back permission");
      control.readerLeaveRegion();
      System.out.println( name + " has handed back permission");

      waitAMoment();
    }
```



```
System.out.println( name + " wants permission to enter region" );
control.writerEnterRegion();
System.out.println( name + " got permission to enter region" );

System.out.println( name + " enters region" );
region.writeStuff( );
waitAMoment();
System.out.println( name + " leaves region" );

System.out.println( name + " wants to hand back permission");
control.writerLeaveRegion();
System.out.println( name + " has handed back permission");

waitAMoment();
```



```
public class Demo
{
    public static void main( String argv[] )
        throws InterruptedException
    {
        final int MAX_READER = 10;

        SomeCriticalRegion region   = new SomeCriticalRegion();
        RegionController   rc       = new RegionController();
        Reader             reader[] = new Reader[MAX_READER];
        Writer             writer   = new Writer( "writer", region, rc );

        for( int counter = 0 ; counter < MAX_READER ; counter++ ){
            reader[counter] = new Reader( "Reader " + counter, region, rc );
            System.out.println( "Reader " + counter + " starting to run" );
            reader[counter].start();
        }
        System.out.println( "Writer starting to run" );
        writer.start();

    }
}
```

```
private Semaphore controllerAccess   = new Semaphore(1);
private Semaphore readersWaiting     = new Semaphore(0);
private Semaphore writerWaiting      = new Semaphore(0);
private Semaphore writerAccess       = new Semaphore(1);

private boolean   writerInsideRegion = false;
private int       nrOfReadersWaiting = 0;
private int       nrOfReadersInside  = 0;
```



```
public void readerEnterRegion( )
{
    try{
        controllerAccess.acquire();
        if( writerInsideRegion ){
            nrOfReadersWaiting++;
            controllerAccess.release();
            readersWaiting.acquire();
        } else {
            nrOfReadersInside++;
            controllerAccess.release();
        }
    } catch (InterruptedException ie ){
        ie.printStackTrace();
    }
}
```



```
public void readerLeaveRegion( )
{
    try{
        controllerAccess.acquire();
        nrOfReadersInside--;
        if( nrOfReadersInside == 0 && writerInsideRegion ){
            writerWaiting.release();
        }
        controllerAccess.release();
    } catch (InterruptedException ie ){
        ie.printStackTrace();
    }
}
```



```
public void writerEnterRegion( )
{
    try{
        writerAccess.acquire();
        controllerAccess.acquire();
        writerInsideRegion = true;
        if( nrOfReadersInside > 0 ){
            controllerAccess.release();
            writerWaiting.acquire();
        } else {
            controllerAccess.release();
        }
    } catch (InterruptedException ie ){
        ie.printStackTrace();
    }
}
```



```
public void writerLeaveRegion( )
{
    try{
        controllerAccess.acquire();
        writerInsideRegion = false;
        if( nrOfReadersWaiting > 0 ){
            nrOfReadersInside  = nrOfReadersWaiting;
            nrOfReadersWaiting = 0;
            readersWaiting.release(nrOfReadersInside);
        }
        writerAccess.release();
        controllerAccess.release();
    } catch (InterruptedException ie ){
        ie.printStackTrace();
    }
}
```



```
Reader 5 has handed back permission
writer wants permission to enter region
Reader 2 leaves region
Reader 2 wants to hand back permission
Reader 2 has handed back permission
Reader 0 wants permission to enter region
Reader 4 wants permission to enter region
writer got permission to enter region
writer enters region
Reader 9 wants permission to enter region
Reader 5 wants permission to enter region
Reader 7 wants permission to enter region
Reader 6 wants permission to enter region
Reader 1 wants permission to enter region
writer leaves region
writer wants to hand back permission
writer has handed back permission
Reader 0 got permission to enter region
Reader 0 enters region
Reader 4 got permission to enter region
Reader 4 enters region
Reader 2 got permission to enter region
Reader 2 enters region
Reader 9 got permission to enter region
Reader 9 enters region
```

Java.util.concurrent

- Innehåller fler klasser för att hantera flertrådade program
- Datastrukturer
  - ConcurrentHashMap
  - ArrayBlockinQueue
    - Buffer -producent-konsument
  - CopyOnWriteArrayList
    - Kopierar listan vid en skrivoperation
  - mm.

Java.util.concurrent.atomic

- Innehåller datatyper AtomicBoolean, AtomicLong, AtomicInteger osv som kan uppdateras "atomically"
- I klasserna finns metoder för att hämta och ändra värdena på ett säkert sätt från flera olika trådar

java.util.concurrent.locks

- Innehåller bla en ReadWriteLock klass