

DD2385

Programutvecklingsteknik

Några bilder till föreläsning 9

6/5 2013

Innehåll

- ▶ Trådar i Java
- ▶ swing- klassen Timer
- ▶ Klient-Server-program

Trådar

- ▶ Tråd = enkel process
- ▶ Ett program kan ha flera trådar, flera aktiviteter pågår "samtidigt" i programmet.
- ▶ Javaprogram utan grafik har **minst en tråd** (**maintråden**)
- ▶ Javaprogram med grafik har **minst två trådar** (**maintråden och grafiktråden**)
- ▶ Man kan skapa egna trådar

När behövs egna trådar ?

Tung bakgrundsaktivitet t.ex.

hämta stor bild, spela musik, beräkning

Aktiva objekt t.ex.

klocka, klientanslutningar till en server

producent/konsument – tillämpningar

Animering, simulering

1) extends Thread

Hur gör man?

Alternativ:

1. Ärv från klassen `Thread`
2. Implementera gränssnittet `Runnable`
(när klassen måste ärvas från annan klass)
3. Enkla animeringar:
använd `javax.swing.Timer`

```
class Active extends Thread {
    Active() {
        // start the Thread,
        // can be done later
        start();
    }

    public void run() {
        // Define the Threads activity
        // There is usually a loop here
    }
}
```

Några metoder i Thread

`run()` trådens aktivitet
`start()` starta, `run()` körs
`sleep(ms)` vänta
`interrupt()` tråden avbryts, kan ej återstarta
`join()` invänta trådens slut

Litet trådexempel

Ett aktivt objekt skriver ut ett meddelande med visst tidsintervall.

```
class Active extends Thread {
    String text; // the message
    int pause;
    int n; // to count the printings
    Active(String t, int p) {
        text = t;
        pause = p;
        start(); // start the thread
    }

    public void run() {
        :
        // see next page
    }
}
```

Litet trådexempel,

fortsättning.

```
class Active extends Thread {  
  
    // as previous page  
  
    public void run() {  
        while (n < 10) { // max 10 printings  
            try {  
                Thread.sleep(pause*1000);  
            }  
            catch (InterruptedException ie){}  
            System.out.println(text + " " + n);  
            n++;  
        }  
        System.out.println(text + " finished");  
    }  
}
```

Att avbryta en tråd

```
class Active extends Thread {  
    boolean keepRunning = true;  
    ;  
    public void run() {  
        while (keepRunning) {  
            try { Thread.sleep(pause*1000);}  
            catch (InterruptedException ie)  
            { keepRunning = false; }  
            System.out.println(text);  
        }  
        System.out.println(text + " was interrupted");  
    }  
}  
:  
Active theThread = new Active(...);  
...  
theThread.interrupt();
```

2) implements Runnable

```
class Active implements Runnable {  
    Thread activity;  
  
    Active() {  
  
        // Create Thread, connect to this object  
        activity = new Thread(this);  
  
        activity.start(); // now or later  
    }  
  
    public void run() {  
        // Define the Threads activity  
        // There is usually a loop here  
    }  
}
```

3) javax.swing.Timer

har konstruktorhuvud

```
Timer (int delay ,  
        ActionListener listener)
```

kommer att upprepa anropet

```
listener.actionPerformed (...)
```

med tidsintervallet delay.

Några metoder i javax.swing.Timer:

```
stop()    start()    setDelay(delay)
```

Tråd(o)säkerhet

Klassiskt exempel

```
class Konto {  
    private int saldo;  
  
    public void taUt(int b){  
        saldo = saldo - b;  
    }  
}
```

Om två trådar samtidigt vill ta ut pengar från samma konto kan det bli problem.

Lösning

Klassiskt exempel

```
class Konto {  
    private int saldo;  
  
    public synchronized void taUt(int b){  
        saldo = saldo - b;  
    }  
}
```

synchronized ⇒ **Konto-objektet låses**
medan metoden **taUt()** exekverar.

Inga andra trådar kan köra synkroniserade metoder på objektet medan **taUt()** exekverar

Singleton nr 2 är ej trådsäker

Skapar objektet i en metod, två trådar kan anropa samtidigt. Lägg till **synchronized**

```
public class Singleton {  
    private static Singleton theInstance = null;  
    private Singleton () {}  
    public synchronized static Singleton  
        getInstance() {  
        if (theInstance == null)  
            theInstance = new Singleton();  
        return theInstance;  
    }  
}
```

Program som kommunicerar

Viktiga begrepp:

- ▶ **IP-adresser** - t.ex. gru-ld03.csc.kth.se eller 130.237.224.131
- ▶ **Portar** – logiska anslutningspunkter till dator, numrerade 0...65535. 0...1024 är reserverade:
 - ▶ 13 för datum/tid
 - ▶ 21 för ftp
 - ▶ 23 för telnet
 - ▶ 80 för HTTPÖvriga nummer får användas fritt
- ▶ **Sockets** - själva anslutningen med möjlighet att ta emot och sända data.

Client-Server

Uppkopplad förbindelse

- ▶ **Serverprogram** - program som väntar på att bli kontaktat av klienter
- ▶ **Klientprogram** - program som kopplar upp sig
- ▶ **Sockets** - en socket i varje ände av förbindelsen
I Java: **Socket** hos klienten, **ServerSocket** hos servern
- ▶ **Strömmar** - Varje socket har en utström och en inström

Klientens uppkoppling, ip-adr och portnr kända

```
try{
    Socket sock =
        new Socket(ipAdress , portnr);
}
catch (IOException e) {
    System.out.println("Ingen anslutning");
}
```

Strömmar för kommunikation

```
InputStream in = sock.getInputStream();
OutputStream out = sock.getOutputStream();
out.flush() krävs för sändning
```

SSP-servern för labb 3

- ▶ En ServerSocket väntar på att klienter ska ansluta
- ▶ En tråd skapas och startas för varje klient.
- ▶ Tråden är ett objekt av
`class Klienthanterare extends Thread`

SSP-servern för labb 3

```
import java.net.*;
import java.io.*;
import java.util.*;
public class Server4712 {
    public static void main(String[] a) {
        try {
            ServerSocket sock =
                new ServerSocket(4712,100);
            while (true)
                new Klienthanterare(sock.accept()).start();
        }
        catch (IOException ioe) {
            System.out.println(ioe);
        }
    }
}
```

SSP-servern, Klienthanterare-klassen

```
class Klienthanterare extends Thread {
    static int antaltradar=0;
    BufferedReader in;
    PrintWriter ut;
    public Klienthanterare(Socket socket){
        try {
            in=new BufferedReader(
                new InputStreamReader(
                    socket.getInputStream()));
            ut=new PrintWriter(
                socket.getOutputStream());
        }
        catch(IOException e){System.err.println(e);}
    }
    // metoden run finns här
}
```

metoden run() i Klienthanterare

```
public void run() {
    Random random=new Random();
    String [] hand={"STEN","SAX","PASE"};
    try {
        // halsningar
        while(true) {
            String indata = in.readLine();
            if(indata==null || indata.equals(""))
                break;
            ut.println(hand[random.nextInt(3)]);
            ut.flush();
        }
        // avslutning
    }
    catch(Exception e) {
        System.err.println(e);
    }
}
```

delar av run() i Klienthanterare

```
// halsningarna
String namn=in.readLine();
System.out.println
    ((++antaltradar)+" : "+namn);
ut.println("Hej, "+namn);
ut.flush();

// avslutning
System.out.println("Nu avslutade "+namn);
antaltradar--;
```

Producent-konsument-tillämpning

- ▶ Producenterna är aktiva objekt (trådar)
- ▶ Konsumenterna är aktiva objekt (trådar)
- ▶ Ett kontrollobjekt har synkroniserade metoder som anropas av producent och konsument
- ▶ Producenter och konsument läggs i vänteläge resp. återaktiveras via kontrollobjektet

Exempel: Våffelkalas

- Producent: `Vaffelgraddare extends Thread`
- Konsumenter: `Vaffelatare extends Thread`
- Kontrollobjekt: `Upplaggsfat`, rymmer **en enda** våffla.
- När en våffla ligger på fatet får producenten vänta
- När fatet är tomt kan producenten lägga dit våffla
- När fatet är tomt får konsumenterna vänta
- När en våffla ligger på fatet kan en konsument ta den
- Gräddaren lägger våfflor **1,2,3, ... 20** på fatet.
Ätarna tar dem!
- All kod läggs på kurshemsidan: `Vaffelgraddare`,
`Vaffelatare`, `Upplaggsfat`, `Vaffelkalas`

```
public class Upplaggsfat {
    private int vaffla=0;
    private boolean vafflorSlut = false;

    public boolean vafflorSlut () {
        return vafflorSlut;
    }

    public synchronized void lagg(int nyVaffla){
        while (vaffla > 0) {
            try{
                wait();
            } catch (Exception e){};
        }
        vaffla = nyVaffla;
        notify();
    }

    // fortsättning följer
}
```

```
public synchronized int tag(){
    while (vaffla==0 && !vafflorSlut()) {
        try{
            wait();
        } catch (Exception e){};
    }
    int tagenVaffla = vaffla;
    vaffla=0;
    notify();
    if (tagenVaffla == Vaffelgraddare.ANTAL_VAFFLOR)
        vafflorSlut = true;
    return tagenVaffla;
}
```

Viktiga metoder

- `wait()`, från `Object`, aktuell tråd läggs i vänteläge
- `notify()`, från `Object`, någon av de väntande trådarna startar igen
- `synchronized` **nödvändigt**, annars kan flera ta samma våffla