Avdeling for informatikk og e-læring, Høgskolen i Sør-Trøndelag

Trådprogrammering

Tomas Holt

17.09.2015

Lærestoffet er utviklet for faget IFUD1042 Applikasjonsutvikling for mobile enheter

Resymé: Denne leksjonen omhandler hvordan man gjøre flere ting "samtidig" ved hjelp av tråder. Tråder er viktige i forbindelse med blant annet nettverksprogrammering.

Contents

6 Trådprogrammering

6.1 Om leksjonen

Vi starter denne leksjonen med å se nærmere på tråder, samt klassen Handler. Boka omhandler tråder i kapittel 23. Vi prøver imidlertid å supplere med en en noe mer detaljert oversikt i denne leksjonen. Klassen Thread blir ikke omtalt i boka, men får mer plass her slik at vi er i stand til å bruke klassen fornuftig. Mer om klassen Handler finner du i kapittel 22 i boka, samt i eksemplet i kapittel 1.6 i denne leksjonen.

Når vi er ferdig med tråder skal vi bruke det vi har lært i forbindelse med nettverksprogrammering (egen leksjon).

6.2 Tråder

Allerede har boka flere ganger nevnt begrepet tråd (eng; thread). I forbindelse med blant annet nettverksprogrammering blir trådbegrepet viktig. Et Java-program kan inneholde flere tråder. Hver slik tråd kjører uavhengig av de andre trådene. Trådene kan nesten sees på som to ulike programmer som kjører samtidig, men i dette tilfellet har de tilgang til de samme dataene. Vi trenger altså tråder i de tilfellene vi vil gjøre flere ting på en gang.

Tenk deg at du lager en applikasjon som skriver ut klokka. For hvert sekund så oppdateres klokka, slik at den hele tiden er riktig. Pseudokoden kan være slik:

```
start..
while (true){
   skriv ut klokka
   sov 1 sekund
}
```

Programmet vil her gå i evig løkke og skrive ut klokkeslettet. Hva om vi ønsker å gjøre noe mer? La oss si at programmet vårt ved hvert hele time skal sende ut en sms. Pseudokoden kan nå bli slik:

```
start..
while (true){
   skriv ut klokka
   sendSMS() //metoden er slik: if (klokke == heltime) send en sms
   sov 1 sekund
}
```

Koden over vil selvsagt fungere, men du oppdager når du prøver å kjøre koden at metoden sendSMS() faktisk normalt bruker ti sekunder på å gjøre seg ferdig. Resultatet er at ved hver hele time så vil klokka stoppe i eleve sekunder (ti sekunder for å sende sms og ett sekund for å sove). Dette er ikke ønskelig! Problemet ligger i at vi i programmet prøver å gjøre to ting på en gang. Løsningen ligger i å skille de to oppgavene i to tråder som kjører uavhengig av hverandre.

```
start..
lag en tråd som kjører sendSMS()
while (true){
   skriv ut klokka
   sov 1 sekund
}
```

Av koden kan du se at logikken for å sende sms er skilt ut i en egen tråd. Denne vil nå kjøre helt uavhengig av resten av koden. Hvordan lages så denne tråden? Det er faktisk veldig enkelt. Klassen vi lager må enten arve fra Thread eller å lage en klasse som implementerer interfacet Runnable. I begge tilfellene må man implementere en metode som heter run(). Det som ligger i run()-metoden vil kjøres i en egen tråd. Trådklassen kan se slik ut:

```
public class SMSThread extends Thread{
   public void run(){
```

```
      while (true){
         if (klokka er heltime) send en sms
      }
   }
}
```

Koden vil nå kunne se slik ut:

```
start..
SMSThread smsThread = new SMSThread();
smsThread.start();//her startes tråden. Merk! Ikke kall run() direkte!
while (true){
   skriv ut klokka
   sov 1 sekund
}
```

Merk hvordan tråden smsThread startes via metoden smsThread.start(). Dette sørger for å starte en ny tråd og kjøre innholdet i run()-metoden. Om man kaller run()-metoden direkte (i stedet for start()) så vil metoden kjøres i samme tråd som kallet kommer fra! Mao. så vil man ikke få en ny tråd i dette tilfellet.

La oss si at vi legger koden over i onCreate()-metoden til en Activity-klasse. Litt mer detaljert kan da koden være slik:

```
public class EksempelAktivitet extends Activity{
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      TextView timeView = (TextView)findViewById(R.id.clock);//finner UI komponent

      SMSThread smsThread = new SMSThread();
      smsThread.start();//her startes tråden.

      while (true){
         timeView.setText(...);//skriver ut klokka til UI
         sov 1 sekund
      }
   }

   //flere metoder...
}
```

Koden over vil ikke fungere som tenkt, da vi aldri får skrevet ut klokka på den uthevede linja. Grunnen til det er at vi aldri kommer ut av onCreate()-metoden. Dette er nødvendig for å kunne oppdatere de grafiske komponentene til en aktivitet. Vi kunne omgått problemet ved kun å skrive til loggen i stedet. Dette er selvsagt greit ved uttesting, men vil ikke være greit for en ordentlig applikasjon. For å få eksemplet til å fungere er vi derfor nødt til å lage to tråder. En for å sende SMS og en for å oppdatere klokka. I tillegg vil aktiviteten kjøre sin egen tråd som er der all UI blir oppdatert.

Vedlagt leksjonen finner du TimeActivity og TimeActivityWithThreads. Førstenevnte viser hvordan kjøring av eksemplet vårt blir uten tråder (via å skrive ut til loggen). Man kan der se at utskrift av klokka vil henge mens man sender sms. I TimeActivityWithThreads løses problemet ved å opprette to tråder, og man kan se at denne løsningen fungerer (her skrives det også til UI). Vi kikker nærmere på denne klassen snart.

6.3 Mer om tråder

Merk at to tråder aldri kan kjøre 100% samtidig om man ikke har flere enn en prosessorkjerne. Implementasjonen vil automatisk sørge for å skifte mellom å kjøre de aktive trådene med jevne mellomrom (dette kan variere i ulike implementasjoner). Det er mulig å spesifisere at en tråd skal gi fra seg prosessoren til en annen tråd. Dette gjøres ved å kalle metoden Thread.yield(). Når dette gjøres i run()-metoden vil tråden gi fra seg kjøretid i prosessoren. Et annet alternativ er Thread.sleep(). Denne metoden vil få tråden til å sove en stund.

Når er så en tråd ferdig? Tråden avsluttes i det man er ferdig med run()-metoden. **Merk at en tråd kun kan startes en gang.** Det er altså ikke mulig å gjenbruk en tråd som er ferdig med run()-metoden.

La oss se litt mer på hvordan man kan stoppe en tråd. Aktiviteten bør kunne stoppe tråden som sender ut sms-meldinger om aktiviteten selv blir stoppet/pauset. I API-dokumentasjonen kan vi se at klassen Thread har metoden stop(). Den bør ikke brukes da den ikke er trygg (deprecated). Det man i stedet gjør i de fleste tilfeller er å kalle smsThread.interrupt(). For at dette skal fungere så må vi imidlertid endre litt i run()-metoden til tråden. I eksemplet over hadde vi følgende:

```
public class SMSThread extends Thread{
   public void run(){
      while (true){
         if (klokka == heltime) send en sms
      }
   }
}
```

Den blir nå i stedet slik:

```
public class SMSThread extends Thread{
   public void run(){
      while (!isInterrupted()){
         if (klokka er heltime) send en sms
      }
   }
}
```

Man sjekker altså enkelt om noen ønsker å avslutte tråden via metoden isInterrupted(). Man bør sjekke med jevne mellomrom i tråden om man har blitt ”interrupted”. Det er imidlertid ikke fullt så enkelt som beskrevet her. En tråd kan nemlig være i flere tilstander. Det er to av dem vi skal bite oss merke i. Den ene er ”kjørende” (runnable), mens den andre er ”ventende” (wait/timed_wait). Vi har så langt bare håndtert tilfellet at tråden er kjørende. Hvis tråden er ventende i det man kaller interrupt() på den så må dette håndteres litt annerledes. Hvordan havner så en tråd i ventende tilstand? Thread.sleep() er en metode som vil endre kjørende tråd til ventende. La oss skrive om koden litt:

```
public class SMSThread extends Thread{
   public void run(){
      while (!isInterrupted()){
          send en sms
          try{
```

```
            Thread.sleep(3600*1000);//sov en time
          }catch(InterruptedException e){
              interrupt();//interrupt-flagget settes på tråden
          }
      }
   }
}
```

Det er slik at metoder som fører tråden inn i en ventende tilstand "vanligvis" vil kaste InterruptedException om noen kaller interrupt()-metoden mens tråden "venter". **Merk at vi selv (i tråden) må håndtere dette unntaket og kalle interrupt() på nytt for å få satt flagget på tråden (da flagget "nullstilles" ved InterruptedException).**

Samme strategi vil dog ikke være til hjelp ved ved lesing fra strømmer (filer og netverksforbindelser) om metodene som brukes er blokkerende. Dette er f.eks. tilfelle om vi bruker BufferedReader.readLine(). Om vi har følgende kode

```
BufferedReader in = .....
public class SMSThread extends Thread{
   public void run(){
      try{
          while (true){
             String str = in.readLine(); //blokkerer og venter på data fra strøm (kan være mot nettverk el
             //gjør noe med dataene
          }
      }catch(IOException e){
         //avslutter tråd
      }
   }
}
```

I denne tråden går vi i løkke og leser fra en strøm. Her vil tråden blokkeres hver gang vi gjør in.readLine(). For å få tråden ut denne typen blokkering kan vi lukke strømmen man leser fra - via in.close(). Om dette gjøres fra en annen tråd så vil in.readLine() kaste en IOException og tråden avsluttes.

6.4 AsyncTask som alternativ til Thread

I Android har man laget et alternativ til Thread-klassen (og Runnable). Denne klassen heter AsyncTask. Hvorfor har man så gjort det? Hovedgrunnen til det er at trådprogrammering kan være litt vanskelig og at man derfor har prøvd å forenkle litt. Det er slik at kun aktivitetens tråd kan oppdatere det grafiske grensesnittet (UI). Om man bruker Thread-klassen så kan man omgå denne begrensningen med å innføre en klasse til, kalt Handler. Vi skal se på dette senere. Med AsyncTask derimot har man laget en løsning som gjør det enklere(?) å påvirke UI direkte. La oss se på klassen:

```
public class AsyncDemo extends AsyncTask<Params, Progress, Result> {
   /* This method is run in the UI thread before starting thread */
   protected void onPreExecute() {}

   /* This is where the work is done. This method is run as a seperate thread (not part of UI thread). */
   protected Result doInBackground(Params... v) {}
```

```
   /* This method is run in the UI thread. Updates of the UI should be done here */
   protected void onProgressUpdate(Progress... values) {}

   /* This method is run in the UI thread when doInBackground is finished */
   protected void onPostExecute(Result result) {}

   /* This method is called when another thread calls AsyncDemo.cancel();
      Method is run in UI thread */
   protected void onCancelled() {}
}
```

Vi kan se at vår klasse arver fra AsyncTask<Params,Progress,Result>. Params, Progress og Result forteller hhv. hvilken type argument som skal inn i doInBackground(), onProgressUpdate() og onPostExecute(), samt returverdi fra doInBackground(). Argumenttypene kan vi altså selv endre etter behov og kan f.eks. være <Void, String, Boolean>.

Metoden doInBackground() kan sammenliknes med run() for Thread-klassen. Det er denne som startes i en ny tråd. De andre metodene kjøres i UI-/aktivitetstråden. Dette er altså metoder som automatisk blir kalt av aktiviteten. Dette gjør at vi kan legge kode i disse metodene som oppdaterer UI. For å starte tråden kalles metoden execute() (i stedet for start()) og avslutting gjøres via cancel() (i stedet for interrupt()).

For mer informasjon se [Android AsyncTask].

6.5 Anonyme klasser

Bruk av anonyme klasser kan være hensiktsmessig i en del sammenhenger. Du kan f.eks. lage en anonym trådklasse og starte tråden inne i en metode:

```
public void makeThread(){
   new Thread(){
       @Override
       public void run(){
          //do something ex. I/O
       }
   }.start();  //start tråden
}
```

Merk at trådklassen i sin helhet er beskrevet inne i metoden. Du kan også se liknende bruk i setSmsView()-metoden i neste kapittel.

6.6 Ferdig eksempelkode

Vi har nå de byggeklossene vi trenger for å kode eksemplet vårt. La oss først se på selve aktiviteten.

```
1  package leksjon.traad;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.os.Handler;
6  import android.view.Window;
7  import android.widget.TextView;
8
9  public class TimeActivityWithThreads extends Activity {
10         private TextView timeView;
11         private TextView smsView;
12         private TimeThread timeThread;//TimeThread extends AsyncTask
13         private SMSThread smsThread;//SMSThread extends Thread
14
15         private Handler handler = new Handler();
16
17
18     @Override
19         public void onCreate(Bundle savedInstanceState) {
20                 super.onCreate(savedInstanceState);
21                 requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);//for
                       showing progress bar
22                 setContentView(R.layout.main);
23                 timeView = (TextView)findViewById(R.id.clock);
24                 smsView = (TextView)findViewById(R.id.sms);
25         }
26
27     @Override
28     /* This method is run when starting and resuming the activity */
29         protected void onResume() {
30                 super.onResume();
31                 smsThread = new SMSThread(this);
32                 smsThread.start();
33
34                 timeThread = new TimeThread(this);
35                 timeThread.execute();
36         };
37
38     @Override
39     /* This method is run when stopping or pausing the activity.
40      * We must stop threads or they will continue to run! */
41     protected void onPause() {
42         super.onPause();
43         timeThread.cancel(true);//stop thread, AsyncTask uses cancel() not interrupt()
44         smsThread.interrupt();//stop thread via interrupt (must handle this in thread).
45     };
46
47      /* This method will be called from the UI thread
48      * (even though the call is done from within the timeThread object).
49      * We can access the timeView component directly. */
50     public void setTimeView(final String text){
51         timeView.setText(text);//Change UI view
52     }
53
54     /* This method will be called from another thread (smsThread).
55      * Don't access the smsView component directly.
56      * Must be accessed via handler!!   */
57     public void setSmsView(final String text){
58         handler.post( new Thread(){
59                         @Override
60                         public void run(){
61                                 smsView.setText(text);//Change UI view
```

```
62                          }
63                  });
64      }
65  }
```

La oss gå litt nærmere inn på koden. I toppen deklarerer vi to tekstfelt (TextView) og to tråder, av typen TimeThread og SMSThread. Disse arver hhv. fra AsyncTask og Thread. onCreate() er temmelig enkel, da vi hovedsaklig henter tekstfeltene fra layout-fila. I onResume() oppretter vi så trådobjektene og starter disse (en for sending av sms og en for å oppdatere klokka). Dette gjøres i onResume(), i stedet for onCreate(), da førstnevnte kjøres både når aktiviteten startes og når den restartes (etter en pause). I onPause() (som blir kalt når aktiviteten må vike) må vi sørge for å stoppe trådene våre. Om vi ikke gjør det vil disse fortsette å kjøre som før! Til slutt har vi to metoder for å oppdatere UI (klokka og statusfeltet for sms). Disse metodene blir altså kalt av trådklassene. La oss se på TimeThread-klassen.

```
1   package leksjon.traad;
2
3   import java.util.Calendar;
4
5   import android.os.AsyncTask;
6   import android.util.Log;
7
8   public class TimeThread extends AsyncTask<Void, String, Boolean> {
9           private TimeActivityWithThreads activity;
10
11          public TimeThread(TimeActivityWithThreads activity){
12                  this.activity = activity;
13          }
14
15          protected void onPreExecute() {
16                  activity.setProgressBarIndeterminateVisibility(true);//show progress
                        bar
17          }
18
19      @SuppressWarnings("unchecked")
20          @Override
21       /* This is where the work is done. This method is run as a seperate thread (not
             part of UI thread). */
22          protected Boolean doInBackground(Void... v) {
23                  while(!this.isCancelled()){
24                          String time = getTime(Calendar.getInstance());
25                          publishProgress(time);//publish new time to UI thread via
                                onProgressUpdate()
26
27                          try{
28                                  Thread.sleep(1000);
29                          }catch(InterruptedException e){
30                                  //don't have to do anything
31                          }
32                  }
33                  return true;
34      }
35
36       @Override
37       /* This method is run by the UI thread. Updates of the UI should be done here */
38          protected void onProgressUpdate(String... values) {
39                  activity.setTimeView(values[0]);
40          }
```

```
41
42      @Override
43      /* This method is run by the UI thread when doInBackground is finished */
44          protected void onPostExecute(Boolean result) {
45                  activity.setProgressBarIndeterminateVisibility(false);
46          }
47
48      @Override
49          protected void onCancelled() {
50                  activity.setProgressBarIndeterminateVisibility(false);
51      }
52
53      /* used to make formatted time string */
54          private String getTime(Calendar cal){
55                  int hour                = cal.get(Calendar.HOUR_OF_DAY);
56                  int minute              = cal.get(Calendar.MINUTE);
57                  int second              = cal.get(Calendar.SECOND);
58
59                  String sHour    = String.valueOf(hour);
60                  String sMinute  = String.valueOf(minute);
61                  String sSecond  = String.valueOf(second);
62
63                  if (sHour.length() == 1)        sHour   = "0" + sHour;
64                  if (sMinute.length() == 1)      sMinute = "0" + sMinute;
65                  if (sSecond.length() == 1)      sSecond = "0" + sSecond;
66                  return sHour + ":" + sMinute + ":" + sSecond;
67          }
68  }
```

I konstruktoren tar vi vare på referansen til aktiviteten vår. Dette gjøres for å kunne oppdatere klokka (UI) når det er tid for det. I onPreExecute() som kjører i UI-tråden (og som kalles **før** tråden startes) starter vi en ”progress bar”. Dette vil vises som en ”roterende framdriftsindikator” øverst til høyre i UI. I doInBackground() ligger koden som kjøres i en egen tråd. Vi går i løkke til vi blir stoppet (via cancel()) og henter ut ny tid (via getTime() - en egendefinert metode). Når vi ønsker å oppdatere UI så kaller vi publishProgress(). Dette metodekallet vil så igjen sørge for å gi ønskede data inn til metoden onProgressUpdate() som igjen vil blir kjørt i UI-tråden. Vi sørger derfor i sistnevnte metode for å kalle aktiviteten sin metode for å oppdatere klokka. Merk også at når vi kaller publishProgress() så tar denne metoden de argumentene som er spesifisert for onProgressUpdate(). I vårt tilfelle er det altså en tabell med strenger. I onPostExecute() og onCancelled() sørger vi for å stoppe ”progress bar”. Merk at onPostExecute() ikke vil kalles om tråden blir avbrutt via cancel().

La oss så se hvordan koden for sending av sms kan være.

```
1   package leksjon.traad;
2
3   public class SMSThread extends Thread{
4           private TimeActivityWithThreads activity;
5
6           public SMSThread(TimeActivityWithThreads activity){
7                   this.activity = activity;
8           }
9
10          @Override
```

```
11          public void run() {
12                  while (!isInterrupted()){ //run until interrupted
13                          sendSMS();
14                  }
15          }
16
17          private void sendSMS(){
18                  activity.setSmsView("Sending new sms");
19
20                  sleep(10);//sleep 10 seconds, pretend sending sms
21
22                  activity.setSmsView("Done sending sms");
23                  sleep(5);//sleep a bit
24          }
25
26          private void sleep(int seconds){
27                  try{
28                          Thread.sleep(seconds*1000);
29                  }catch(InterruptedException e){
30                          Thread.currentThread().interrupt();//set interrupt flag on
                                thread (must do this)!
31                  }
32          }
33  }
```

Vi tar også i dette tilfellet inn referansen til aktiviteten i konstruktoren. Vi tar vare på denne slik at vi kan kalle aktiviteten når vi har behov for å oppdatere UI. I run()-metoden går vi i løkke til vi blir avbrutt med interrupt(). Metoden sendSMS() sørger for å oppdatere UI når man ”starter å sende sms” og oppdaterer UI igjen når ”utsending er ferdig”. Ettersom vi ikke har mulighet til å faktisk sende sms (på emulator) så illustreres dette ved å sove en periode (sleep()).

Du bør nå se på hvordan de to trådene oppdaterer UI via setTimeView() for TimeThread og setSmsView() for SMSThread. Du kan se at setTimeView() oppdaterer aktuelt tekstfelt direkte. Dette kommer av at denne metoden alltid kalles fra UI-tråden. Når det gjelder setSmsView() derimot så kalles denne fra run()-metoden (her egentlig en metode som blir kalt av run()), noe som gjør at denne kalles fra en annen tråd en UI-tråden. Man kan derfor ikke oppdatere ønsket UI-komponent direkte. Det man gjør er i stedet å kalle post()-metoden på et Handler-objekt. Metoden tar så som argument et Thread-objekt, men merk at denne tråden ikke skal startes av oss. Handler-objektet vil nå sørge for at oppdatering (som beskrevet i Thread-objektet) av UI-komponenten skjer på en sikker måte.

Om du skulle prøve å aksessere en UI-komponent fra en annen tråd enn aktivtetens vil du få kjøretidsfeil.

6.7 Referanser

[Android AsyncTask] http://developer.android.com/reference/android/os/AsyncTask.html