Gjennomgang av programvaren

Her finnes en omtale av programmeringsomgivelsen som skal brukes, samt en introduksjon til ulike programmeringsmekanismer som vil trenges.

Programmeringsomgivelse – Oracle JDK

Som i flere tidligere programmeringsemner skal dere også i dette kurset bruke Sun's Java Development Kit versjon 1.8.0.31 til å kompilere og kjøre Javakode. Dere kan skrive kildekode i den editoren dere selv måtte ønske, for eksempel IntelliJ IDEA (gratis studentlisenser), Eclipse, Vim eller Emacs. De kommandoene dere vil få bruk for i JDK er:

- **javac**: Kompilator som kompilerer javakode til såkalt bytecode; dvs. en maskinuavhengig maskinkode. Ingen lenking er nødvendig (det foretas dynamisk). Dere må oppgi hele navnet på filen - inkludert endelsen .java - for at javac skal finne den.
- **java**: Dette programmet brukes for å kjøre applikasjoner. Dere må oppgi hvilken .class fil som skal kjøres. Dere kan også gi inn parametere til applikasjonen, for eksempel slik:
  java MyClass param1 param2 param3
  Parametrene som følger etter klassenavnet gis inn til applikasjonen som args[]-tabellen i main-metoden.

Tråder i Java

Tråder i Java er instanser av klassen Thread, og opprettes og slettes som andre objekter. Etter at man har opprettet en tråd, må man starte kjøringen av den med start()-metoden. Objektet vil da utføre en kodesekvens som en egen tråd. Det er to måter å spesifisere hvilken kode en tråd skal kjøre. For det første kan man subklasse trådklassen og redefinere run()-metoden. Denne metoden blir da kjørt av det nye trådobjektet. Den andre måten er å lage en klasse som implementerer grensesnittet Runnable og dermed metoden run(). En instans av denne klassen sendes da som innverdi til konstruktøren til trådobjektet. Run()-metoden til dette nye objektet vil da bli kjørt av den nye tråden. Når run()-metoden til en tråd avslutter/returnerer, avslutter tråden kjøringen sin. Dette er den anbefalte måten å avslutte en tråd på.

Nå vil metoder i trådklassen som kan være nyttige bli beskrevet. Trådklassen inneholder flere metoder enn de som er beskrevet her. En kan også se 'http://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html' for å se noen eksempler på bruk av klassen.

- **public Thread(**ThreadGroup group, Runnable target, String name**)**
  Konstruktoren til trådobjektet kan ta en eller flere av de tre parametrene som er nevnt over i den rekkefølgen som er angitt:
  - **ThreadGroup group**: Denne parameteren sier hvilken trådgruppe tråden skal knyttes til. Trådgrupper er grupper av tråder som man kan tenke seg å behandle som en enhet. Man kan kalle ulike funksjoner på et helt sett av tråder på en gang ved å bruke trådgrupper. Vanligvis har man ikke behov for å spesifisere noen trådgruppe.
  - **Runnable target**: Denne parameteren brukes hvis man ønsker å få kjørt run() metoden til et Runnable-objekt i stedet for å subklasse trådklassen.
  - **String name**: Brukes til å sette et navn på tråden. Vanligvis ikke nødvendig, men kan gjøre debugging lettere.

- **public static void sleep(**long millis**)**
  Denne metoden fører til at tråden som kjører nå, stopper utførelsen i det oppgitte antall millisekunder eller til den blir avbrutt.

- **public void interrupt()**
  Avbryter en tråd. Dette vil vekke en tråd som sover eller venter på en betingelse. (Se Synkronisering i Java).

- **public static Thread currentThread()**
  Returnerer trådobjektet til den tråden som kjører nå.

public final void suspend()

Denne funksjonen fører til at tråden den blir kalt i, stopper å kjøre. (Anbefales ikke brukt i JDK 1.8). Deprecated

public final void resume()

Hvis denne blir kalt i en tråd som har blitt suspendert, vil den suspenderte tråden fortsette kjøringen. (Anbefales ikke brukt i JDK 1.8). Deprecated

public synchronized void start()

Starter utførelsen av tråden funksjonen blir kalt i.

public final void stop()

Denne funksjonen avslutter en tråd ved å lage et ThreadDeath unntak. Dette unntaket bør ikke fanges. (Anbefales ikke brukt i JDK 1.8). Deprecated

Deprecated metoder

Metodene suspend(), resume() og stop() er deprecated i JDK 1.8, hvilket vil si at de ikke anbefales brukt. Stop() er deprecated fordi et kall til denne metoden mens tråden holder på med å modifisere variable, kan føre til at disse blir satt i en inkonsistent tilstand. I stedet bør man la run()-metoden avslutte hvis man ønsker å avslutte en tråd. Resume() og suspend() er deprecated fordi de kan føre til vranglåser (deadlocks). Man har sjelden bruk for disse funksjonene, og hvis man har det kan man oppnå det samme ved å bruke wait() og notify() som er beskrevet i neste avsnitt. For en mer fullstendig beskrivelse av hva som er galt med suspend(), resume() og stop(), og hvordan man klarer seg uten dem, se http://docs.oracle.com/javase/8/docs/technotes/guides/concurrency/threadPrimitiveDeprecation.html.

Synkronisering i Java

Java bruker en variant av monitorkonseptet hvor alle objekter potensielt kan fungere som monitorer. Man trenger ikke å opprette monitoren eksplisitt, men indikerer i stedet hvilken kode man vil beskytte og hvilket objekts monitor som skal kontrollere tilgang til denne koden. Det er to måter å beskytte kode med en monitor på, og begge bruker ordet synchronized.

Den første måten er å deklarere en metode som synchronized. Metoder som er synchronized beskyttes av monitoren til objektet metoden kjøres på. Hvis alle metodene i en klasse er synchronized har man en klassisk monitor. Den andre måten å beskytte kode på er å lage en synkronisert blokk med kode. Dette gjøres på følgende måte:

```
synchronized(objekt) {
   // kode som skal være beskyttet
}
```

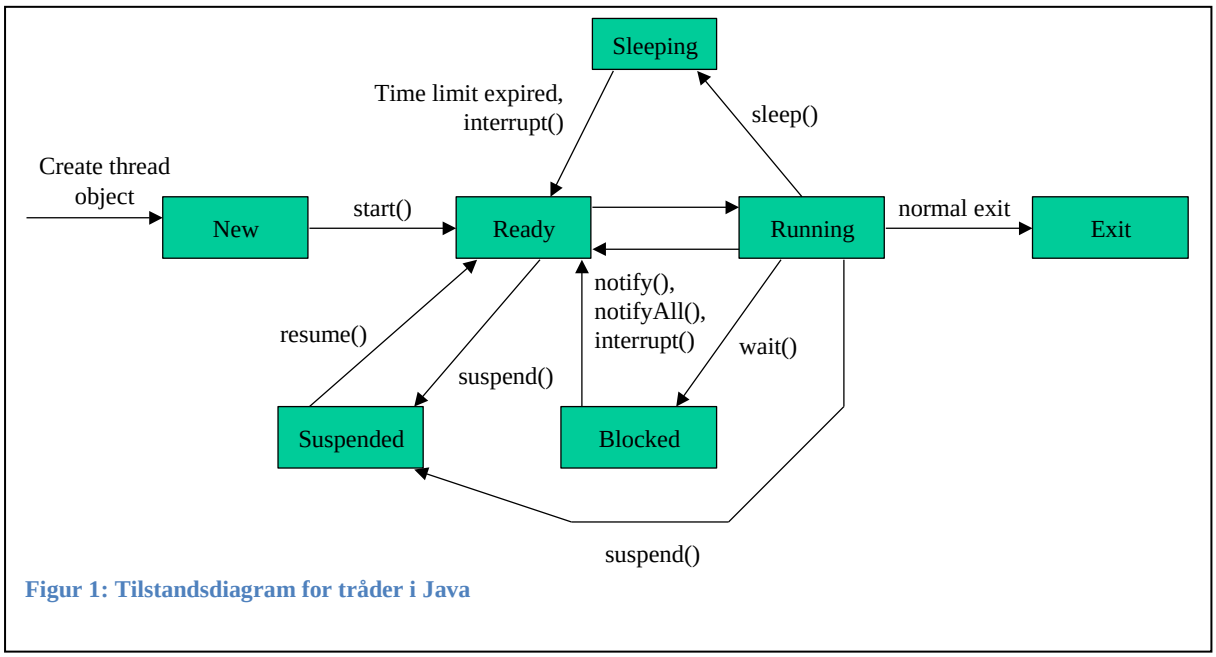
Koden inne i synchronized-blokka er beskyttet av monitoren til objektet inne i parentesene.

Når en tråd ønsker å kjøre kode/metoder som er synchronized, må tråden prøve å få monitoren (låsen) som beskytter koden. Hvis monitoren er ledig, får tråden låsen, og kan kjøre koden. Når tråden er ferdig med koden som er synchronized, slipper den låsen igjen slik at andre tråder får lov til å kjøre koden. Hvis en tråd prøver å kjøre kode som er synchronized, og låsen er opptatt, må tråden vente til låsen blir frigjort før den kan fortsette. På denne måten oppnår man gjensidig utelukkelse.

En tråd kan godt ha monitoren til flere objekter samtidig, og den kan også ha monitoren til det samme objektet flere ganger. Det siste skjer hvis man i en synchronized metode eller kodebit kaller en annen synchronized metode i det samme objektet. Siden objektet allerede har monitoren, vil dette gå bra, men tråden må slippe opp monitoren like mange ganger som den har låst den før andre tråder kan slippe til.

Hvis en tråd oppdager at den ikke har en ressurs som den trenger inne i en synkronisert kodebit, kan den kalle wait()-metoden til objektet som den har monitoren til. Denne funksjonen tilsvarer cwait() i læreboka, bortsett fra at det bare er én ventekø. Når en tråd kaller wait() vil tråden settes til å vente, og slipper monitoren. Hvis en tråd har lagt til en ny ressurs, kan den kalle notify()-metoden til objektet den har monitoren til. Denne funksjonen tilsvarer cnotify() i læreboka, og vil vekke en tilfeldig tråd som har kalt wait()-metoden til objektet notify() ble kalt på. Man kan også vekke opp alle trådene som har kalt wait() på et objekt ved å kalle notifyAll() på dette objektet. Denne funksjonen tilsvarer cbroadcast() fra læreboka. En tråd vil også bli vekket opp fra wait() hvis den blir avbrutt av et interrupt()-kall. Tråder som blir vekket fra en wait() på grunn av et kall til notify(), notifyAll() eller interrupt() vil måtte konkurrere om å få tak i monitoren til objektet på nytt før de kan fortsette kjøringen sin. Dette sikrer at maksimalt én tråd til enhver tid kjører kode som er synchronized.

Funksjonene wait(), notify() og notifyAll() er implementert i Object klassen og kan dermed brukes i alle klasser, også egendefinerte. Alle Javaklasser arver fra Object enten direkte eller indirekte. En klasse som ikke arver noe som helst har en implisitt ”extends java.lang.Object” satt inn. Alle klasser som arver fra en annen klasse vil arve funksjonene fra Object gjennom den.



Figur 1: Tilstandsdiagram for tråder i Java

Figur 1 viser hvordan de ulike funksjonskallene påvirker tilstanden til en tråd i Java. Stop() er ikke vist - og avslutter en tråd uansett hvilken tilstand den er i.

Eksempel på synkronisering

Her er et enkelt og trivielt eksempel på en applikasjon med to tråder og et behov for gjensidig utelukkelse. Begge trådene skriver ting til skjermen, men kun en av dem skal kunne skrive til skjermen på en gang. Dette løses ved at en tråd må ha monitoren til Skjerm-objektet for å kunne skrive til skjerm.

Skriver.java

```
public class Skriver extends Thread
{
   // To ord som skal skrives til skjermen av denne skriveren
   private String forsteOrd, andreOrd;
   // Referanse til Skjerm-objektet
   private Skjerm skjerm;

   // Oppretter et nytt Skriver-objekt
   public Skriver(Skjerm skjerm, String forsteOrd, String andreOrd) {
      super();
      this.skjerm = skjerm;
      this.forsteOrd = forsteOrd;
      this.andreOrd = andreOrd;
   }

   // Metoden som utføres av denne Skriver-tråden
   public void run() {
      for(int i = 0; i < 5; i++) {
         // Skriv ut de to ordene
         skjerm.skrivUt(forsteOrd, andreOrd);
         // Vent noen tidels sekunder
         try {
            Thread.sleep((int)(Math.random()*700));
         } catch (InterruptedException e) {
            // Kommer hit hvis noen kalte interrupt() på denne tråden.
            // Det skjer ikke i denne applikasjonen.
         }
      }
   }

   // Metoden som blir kjørt når applikasjonen starter
   public static void main(String[] args) {
      // Opprett skjerm-objektet
      Skjerm skjerm = new Skjerm();
      // Start to skrivertråder som skriver ut ordene som ble gitt inn fra kommandolinja
      Skriver skriver1, skriver2;
      skriver1 = new Skriver(skjerm, args[0], args[1]);
      skriver1.start();
      skriver2 = new Skriver(skjerm, args[2], args[3]);
      skriver2.start();
   }
}
```

Skjerm.java

```
public class Skjerm
{
   // Siden denne funksjonen er synchronized,
   // kan bare en tråd være inne i den av gangen
   public synchronized void skrivUt(String ord1, String ord2) {
      System.out.print(ord1);
      try {
         Thread.sleep(100);
      } catch (InterruptedException e) {}
      System.out.println(ord2);
   }
}
```

Resultat av kjøringen

Her er en utskrift fra kjøring av eksempelapplikasjonen:

```
java Skriver kake spade bamse mums
kakespade
bamsemums
kakespade
bamsemums
bamsemums
kakespade
kakespade
bamsemums
kakespade
bamsemums
```

Og her er en utskrift av det samme programmet, der skrivUt-metoden ikke er synchronized:

```
java Skriver kake spade bamse mums
kakebamsespade
mums
bamsekakemums
spade
kakebamsespade
mums
kakespade
bamsemums
bamsekakemums
spade
```