Övning, Threads

Uppgift 1

Skapa en klass DigitalKlocka, som skriver ut tiden en gång i sekunden. Klassen ska implementera Runnable, och ha en egen tråd som datamedlem.
Det ska vara möjligt att starta en Klocka genom att anropa metoden startClock(), varvid en tråd skapas och startas. Denna metod ska kunna anropas flera gånger för en klocka. Det ska också vara möjligt att stoppa tråden med stopClock(), genom att signalera till run() att returnera. Detta görs enklast genom att sätta referensen till tråden till null, och låta run() kontrollera detta.
Aktuell tid får du enklast m h a klassen java.util.Date.
Skapa ett main där ett några klockor startas och får ticka ett tag, för att sedan avbrytas.[1]

Uppgift 2, synkronisering

Studera t ex klassen ArrayList och fundera över vilka metoder som måste synkroniseras för att göra objekt av klassen trådsäkra. Ett sätt att åstadkomma detta är att kapsla in ett ArrayListobjekt i en ny klass, SynchronisedArrayList, (som lämpligen implementerar List-interfacet) och ge metoderna i synkroniserade versioner.

Notera (från API-doc):
A List can be "wrapped" using the Collections.synchronizedList method. This is best done at creation time, to prevent accidental unsynchronized access to the list:

```
List list = Collections.synchronizedList(new ArrayList(...));
```

It is imperative that the user manually synchronize on the returned list when iterating over it:

```
List list = Collections.synchronizedList(new ArrayList());
    ...
synchronized(list) {
    Iterator i = list.iterator(); // Must be in synchronized block
    while (i.hasNext())
        foo(i.next());
}
```

Uppgift 3, synkronisering

Skriv en klass Queue (m h a LinkedList), som representerar en kö där producenter kan lämna meddelanden, metoden putLast(), och konsumenter hämta meddelanden, metoden getFirst(). Låt kön rymma max 5 meddelanden.
Queue klassen har två synkroniserade metoder:

- putLast() – Anropas av producenter. Metoden kontrollerar i en **loop** om kön är full, i så fall anropas wait(), så att anropande tråd väntar. Om kön inte är full placeras meddelandet sist och notifyAll() anropas så att eventuella konsumenter som väntar efter att ha anropat getFirst() när kön var tom väcks.

[1] Tips: För att få korrekt tidszon använder du dig av klassen java.text.DateFormat. Du får tillgång till ett DateFormat-objekt för en given tidszon med:

```
DateFormat df = DateFormat.getTimeInstance();
df.setTimeZone(TimeZone.getDefault()); // Maskinens inställningar
```

alternativt

```
df.setTimeZone(TimeZone.getTimeZone("Europe/Stockholm")); // "CET", "GMT"
System.out.println( df.format( new Date() ));
```

- getFirst() – Anropas av konsumenter. Metoden kontrollerar i en **loop** om kön är tom, i så fall anropas wait(),så att anropande tråd väntar. Om kön inte är tom hämtas ett meddelande och notifyAll() anropas så att eventuella producenter som väntar efter att ha anropat putLast() för en fylld kö väcks.

Skapa sedan producenter och konsumenter som aktiva objekt (med egna trådar), som med jämna mellanrum anropar putLast() och getFirst(). Huruvida dessa aktiva objekt kan göra det de vill, eller måste vänta, bestäms helt och hållet i de ovan nämnda metoderna.
Skriv ett main där du skapar producenter och konsumenter som manipulerar kön. Studera vad som händer.