Tentamen
Nätverksprogrammering
Del 2

2008–05–26, 8.00–13.00

Tillåtna hjälpmedel för denna del av tentamen:

- Java snabbreferens
- Kursboken: Java Network Programming av Eliotte Rusty Harold.
- Valfri lärobok i Java
- Utskrift av OH-bilder från föreläsningarna.

Denna tentamen i kursen Nätverksprogrammering består av två delar – en del som innehåller frågor av teoretisk/principiell/utredande karaktär och en del som innehåller praktiska programmeringsuppgifter. Detta är del 2. Den ska du ha erhållit tillsammans med ett färgat tentamensomslag när du lämnade in din lösning på del 1 av tentamen.

För godkänt betyg på tentamen krävs sammanlagt minst 20 poäng på tentamen, varav minst 8 poäng på vardera deltentamen. För högre betyg krävs naturligtvis mer, så gör så många uppgifter du kan.

Denna deltentamen innehåller en uppgift uppdelad i tre deluppgifter som kan ge 5, 10, respektive 5 poäng. Deluppgifternas ordningsföljd avspeglar inte nödvändigtvis uppgifternas svårighetsgrad.

---

1. Meddelandehanteringssystem

Uppgiften handlar om att skriva de centrala delarna, server och nätverksgränssnitt, av ett system med vilket distribuerade applikationer kan skicka meddelanden till varandra i form av strängar. Med dess hjälp skulle man till exempel kunna implementera ett enkelt e-postsystem eller utnyttja det som nätverksmotor i i stort sett vilken distribuerad applikation som helst som bygger på utbyte av meddelanden.

Systemet som ska utvecklas består av en komplett server samt en klass, `MessageClient`, som kan användas på klientsidan för att kommunicera med servern (skicka och ta emot meddelanden). Ett meddelande i sig består av en sträng och mottagaren (klienten) av meddelandet identifieras av en unik sträng. Servern håller reda på alla meddelanden som sänts till olika mottagare tills de hämtas av respektive mottagare, varvid de tas bort. Flera meddelanden till en och samma mottagare ska kunna lagras. Om det inte finns några meddelanden till en viss klient ska man kunna vänta tills ett meddelande anländer. Meddelandena i sig ska hanteras på servern av en klass som implementerar interfacet `MessageStore` nedan.

```
interface MessageStore {
   /**
    * Adds a new message to the database.
    * @param receiver Identifies the receiver of the message.
    * @param message The message to send.
    */
   void addMessage(String receiver, String message);
```

```
    /**
     * Retrieves the next message to the indicated receiver. If no message is
     * available for the receiver, the method blocks until a message arrives.
     * @param receiver Identifies the receiver.
     * @return The message.
     */
    String nextMessage(String receiver);
}
```

Förutom ett databasobjekt som implementerar interfacet `MessageStore` behöver man på servern ha programvara som hanterar nätverksuppkopplingar och som knyter ihop klienternas uppkopplingar med databasobjektet.

På klientsidan ska det finnas en klass `MessageClient` som används för att ansluta till servern. Resten av programvaran på klientsidan beror på applikationen och ingår ej i uppgiften. Klassen `MessageClient` har följande externa gränssnitt:

```
class MessageClient {
    /**
     * Constructor.
     * @param server The network address of the server to connect to.
     */
    public MessageClient(String server) { ... }

    /**
     * Send a message to a receiver.
     * @param receiver The receiver of the message.
     * @param message The message to send.
     */
    public void send(String receiver, String message) { ... }

    /**
     * Returns the next message for this client. If no message is available,
     * the method blocks until a message arrive.
     * @param receiver The receiver of the message.
     * @returns The message.
     */
    public String receive(String receiver) { ... }
}
```

a) Skriv en klass `MessageStoreImpl` enligt beskrivningen ovan som implementerar interfacet `MessageStore` och som ansvarar för hanteringen av meddelanden. Tänk på att klassen måste klara av att flera trådar anropar `addMessage`/`nextMessage` samtidigt oberoende av varandra.

(5p)

b) Utgå från klassen `MessageStoreImpl` du skrev i föregående deluppgift (uppgift a) och skriv resten av klasserna som behövs i servern samt klassen `MessageClient` på klientsidan. Använd vanliga TCP-uppkopplingar för att kommunicera mellan klienten och servern. Se till att flera klienter kan ansluta parallellt och att det finns felhantering som gör att en enskild fallerande klient inte medför problem för övriga klienter eller servern i övrigt.

(10p)

c) Gör likadant som i föregående uppgift (dvs implementera servern samt klassen `MessageClient` på klientsidan), men använd RMI för kommunikationen mellan klienten och servern.

(5p)

Slut – Glad sommar!