Laboration 2: Ett kommunikationssystem

1 Syfte

- Att arbeta ännu mer med OO-design och programmering, framför allt programmering mot gränssnitt.
- Undantag och felhantering.
- Parallellism

2 Uppgift

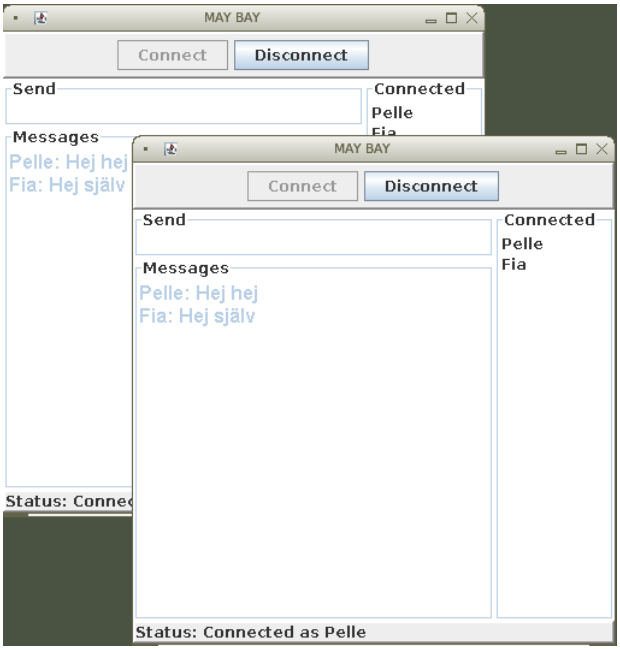
Ni skall implementera ett litet kommunikationssystem bestående av en klient- och en server-applikation. Systemet kan hantera meddelanden (chat) och (enkel) filöverföring mellan användare se figur.

3 Java RMI och Nätverk

Vi kommer att använda Java:s Remote Method Invocation (RMI). RMI gör det möjligt att anropa metoder på objekt i andra JVM:er (potentiellt på andra maskiner). De exakta detaljer om hur RMI fungerar skall vi inte behöva bekymra oss om, i koden ser anropen ut som vanligt (förutom några speciella undantag)[1].

I botten på RMI finns ett TCP/IP-nätverk. Vi måste alltså använda IP-adresser och portar för att tala om vart objekt m.m. finns (Servern är ett objekt). För att förenkla kommer vi att simulera nätverket genom att använda den s.k. "loopback" adressen, IP-nummer 127.0.0.1 (kallas också localhost). Innebär att vi kan köra klienter och servern på samma dator. Se vidare Main.java (servern), TestServer.java (klienten) och filerna runserver.sh och runclient.sh i projektkatalogerna för respektive. OBS! Servern registrerar sig i RMI-registret och klienten söker rätt på servern i registret, klienten skall alltså inte registrera sig.

[1] Förhoppningsvis...RMI har blivit mycket lättare att använda i Java >= 1.5. Se upp med gamla artiklar på nätet om; rmic, stubs, skelletons, e.t.c. Det mesta skall fungera utan att vi skall behöva göra något speciellt.



Figur 1: Två klienter

4 OO-Modellen

Hela modellen är given och finns på server, se paketet core.

5 Funktionalitet

5.1 Klientapplikationen

Klienten kan göra följande (se figur);

- Klienten kan ansluta sig till Servern (Connect). Då klienten är ansluten kan han/hon skicka meddelanden till andra anslutna klienter. Alla anslutna visas i listan till höger (Connected).
- Klienten kan koppla ner sig (Disconnect).
- Meddelanden skickas genom att man skriver något i det tomma textfältet och trycker Enter. Samtliga anslutna får meddelandet (broadcast).
- Då man dubbelklickar på någon ansluten i Connected-listan öppnas ett fönster som visar vilka filer man kan ladda ned från personen. Det som listas är filerna i katalogen upload i projektkatalogen. Om man markerar en fil och klickar Download laddas filen ner (direkt från den andra klienten). Nedladdade filer hamnar i katalogen download.

Klientapplikationerna startas med s.k. skript, se runclient.sh och runclient2.sh (fungerar inte på Windows, isf får du skriva om till .bat filer). Eventuellt måste skripten göras exekverbara. Görs med (i terminalen);

```
$ chmod u+x runclient.sh
```

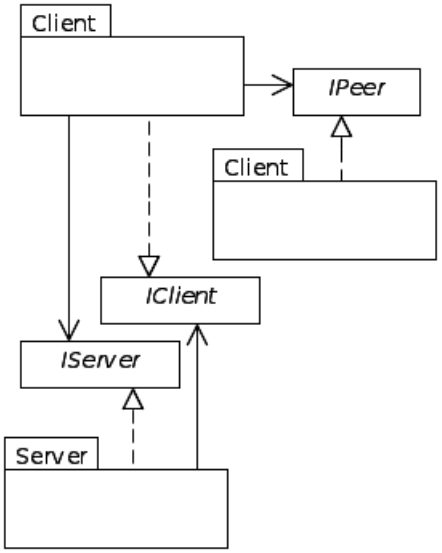
5.2 Serverapplikationen

Grundläggande server-funktionalitet (mer behövs, ingår i uppgiften att reda ut);

- Det finns några registrerade (hårdkodade) klienter i server. Dessa skall kunna logga in och logga ut.
- En klient måste kunna få direkt kontakt med en annan klient (kallas då Peer). Detta därför att all filöverföring skall ske direkt mellan klienter (finns inga filer på servern). Kallas ofta peer2peer (P2P) nätverk.

Servern saknar helt GUI. Det är lämpligt att låta servern logga allt som händer (skriva ut i terminalen), se kod. Även servern startas med ett skript, se runserver.sh

6 Systemarkitektur



Server implementerar gränssnittet IChatServer (IServer i bilden) som används av klientapplikationen. Klienten implementerar IChatClient (IClient i bilden) som används av Servern. IPeer används mellan klienter vid filnedladdning. Samtliga gränssnitt "extends Remote" eftersom metodanropen sker med RMI. Samtliga metoder i gränssnitten måste ha "throws RemoteExcerption".

7 Designmodell för Klienten

En förenklad MVC modell där GUI-delarna direkt anropar Client-klassen. Klassen MainFrame implementerar gränssnittet IObserver för att kunna ta emot uppdateringar från Client, se kod.

- Main-klassen startar applikationen.
- Alla synliga delar av GUI:et är klara, men naturligtvis inte händelsehantering och notifikationer från modell till GUI. All kod i paketet view.
- klassen Client i paketet client.client är implementationen av ett antal gränssnitt. Gränssnitten skall utformas enligt the interface segregation principle. Client kommer att använda State-mönstret, se vidare nedan.
- paketet io hanterar läsning och skrivning av filer.
- paketet exception innehåller en undantagsklass, används för att wrappa checked exceptions och för att höja abstraktionsnivån.
- paketet util innehåller hjälpklasser.

Klienten kan vara i olika tillstånd. Utifrån detta skall klienten använda designmönstret State. Klienten byter alltså tillstånd genom att byta objekt (ändrar referens till annat objekt). Klassen StateContext håller reda på tillstånden (objekten) samt det aktuella tillståndet. RMI hanteringen skall ligga i tillståndsobjekten, inte i Client (klienten startar alltid i nedkopplat läge).

8 Designmodell för Servern

Själva servern består bara av en enda klass service.Server. All RMI-hantering sköts av denna klass, ingen RMI-kod i modellen. Main-klass används för att starta server (den är färdig).

9 Felhantering

Systemet skall tåla att applikationer kraschar, t.ex. skall en klient kunna ansluta sig på nytt efter det att den kraschat. Ett knippe undantag (krascher simuleras med Ctrl c i terminalen man startade applikationen från):

- Klienten kraschar vad gör servern?
- Klient försöker ansluta till server som inte är igång.
- Klient skickar null-parameter till servermetod?
- Server kraschar, vad gör klienten?

Koncentrera all felhantering på klientsidan till Client-klassen (för vidare transport till GUI:et). Skicka alla fel på lägre nivåer uppåt till Client.

10 Parallellism

Några punkter

- Detta är ett parallell system, flera klienter kan samtidigt kontakta servern. Varje RMI-anrop skapar normalt en egen tråd! Se över så att servern är trådsäker.
- Om inkommande anrop på klientsidan skall vidare upp till GUI:et får man byta till Swing-tråden (allt i GUI:et skall ritas av denna). Man får alltså "lämna" över från en tråd till en annan.
- Tidskrävande operationer skall köras i egna trådar använd Swing worker för filnerladdning.
- Servern bör med jämna mellanrum kontrollera om någon klient kraschat. Kontrollen skall köras i en egen tråd.
- Använd högnivåtrådning, SwingWorker och TimerTask och liknande inte klassen Thread!

11 Filöverföring

Filöverföring sker som sagt direkt mellan klienter. All filhantering på låg nivå är klar, se paketet client.io och klassen ChatFile.

12 Övergripande process

Det finns NetBeans-project att ladda ner, ett för klient och ett för server. Båda är körbara.

1. Testa att starta server och klienter m.h.a. skripten. Öppna en terminal för varje skript. För att starta servern skriver man:

   ```
   $ ./runserver.sh
   ```

2. En metod, ping, är implementerad i servern. Det finns en test i klienten som anropar metoden. Starta servern och kör testet för att, om möjligt, få en första kontakt mellan klient och server.

3. Implementera preliminära versioner an någon metod i serverns interface. Testa!

   OBS! Tanken är att ni i fortsättningen själva skall hitta de metoder som behöves i respektive gränssnitt.

4. Gör en tillfällig version av klienten (Client) som använder RMI för att anropa server (RMI koden skall senare flyttas till de olika tillståndsklasserna). Målet är att få ett anrop att gå hela vägen från klient-GUI:et till Servern.

5. Fundera på vad servern måste kunna göra med klienten. Implementera därefter en metod som går "hela varvet", från klient-GUI, till Client, till server och tillbaks till klient-GUI.
6. Lägg till State mönstret för klienten.
7. Lägg till fler metoder.
8. Kontrollera felhantering och parallellism.
9. Implementera filöverföringen.

13 Redovisning

Som tidigare labbar.

Inlämningsdatum Se kurssida.