Laboration 3: Ett kommunikationssystem

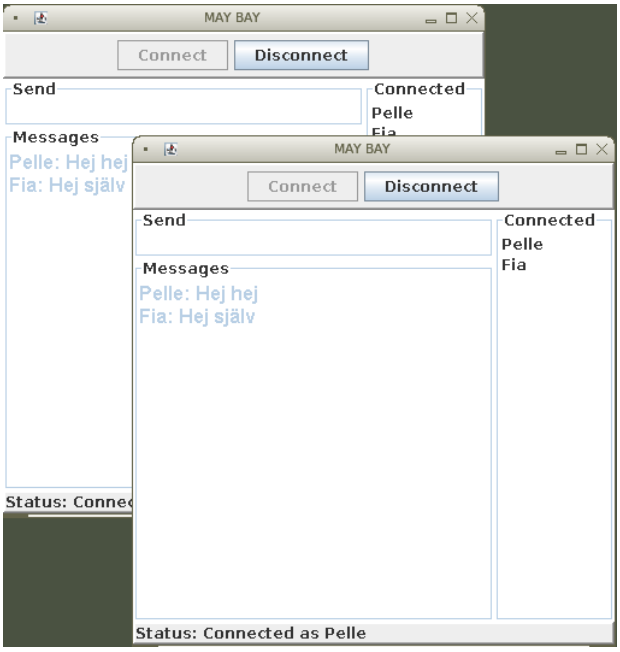
1 Syfte

- Att arbeta ännu mer med OO-design och programmering mot gränssnitt.
- Undantag och felhantering.

Varning Labben bygger på Java 1.6, Java 7 (1.7) verkar inte fungera?

2 Uppgift

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



3 Bakgrund

Eftersom applikationen är distribuerad måste det finnas något sätt att kommunicera (över ett nätverk). För att göra det hela lite enklare kommer vi att köra alla applikationer på samma dator. M.h.a. en s.k. loopback-adress (IP-nummer 127.0.0.1, kallas även "localhost") kan vi simulera anrop över nätet.

Vi kommer att använda Java's Remote Method Invocation (RMI). RMI gör det möjligt att anropa objekt (metoder) i andra JVM:er (potentiellt på andra maskiner). I princip innebär det att objekt kan öppna (normal slumpvisa[1]) portar och ta emot anrop på dessa. De exakta detaljer om hur RMI fungerar skall vi inte behöva bekymmra oss om[2].

4 Funktionalitet

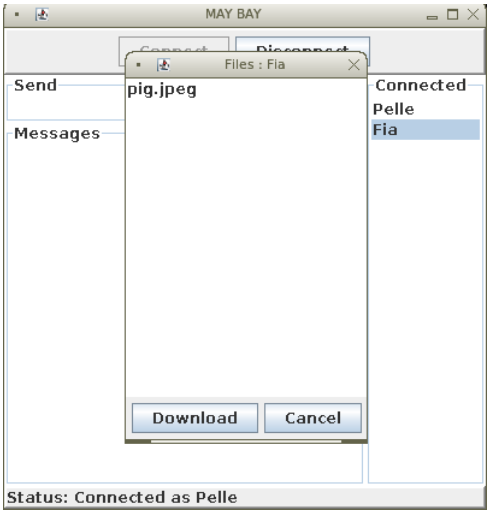
4.1 Klientapplikationen

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 texfältet "Send" och trycker enter. Samtliga anslutna får meddelandet (broadcast).
- Då man dubbelklickar på ett namn i Conneced-listan öppnas ett fönster som visar vilka filer man kan ladda ned från personen. Det som listas är filerna i katalogen upload (finns färdig då du hämtar starkoden). Om man markerar en fil och klickar Download laddas filen ner (direkt från den andra klienten). Nedladdade filer hamnar i katalogen download. Man kan inte ladda ner sina egna filer.

[1] D.v.s. svårt att köra programmet över internet, normal tillåts inte att man öppnar portar hur som helst.

[2] 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.



Klientapplikationerna startas med s.k. skript, se runclient.sh och runclient2.sh (fungerar inte på Windows, isf får du skriva om till .bat filer).

4.2 Serverapplikationen

Grundläggande serverfunktionalitet (mer behövs, ingår i uppgiften att reda ut);

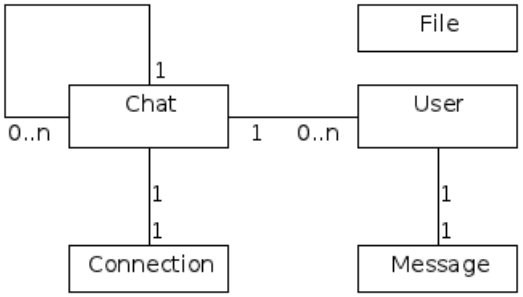
- Klienter skall kunna registrera sig (registrerade klienter läggs i en Map, se utdead kod) och avregistrera sig. Alla klienter måste registrera sig under ett unikt användarnamn (lösenord saknas)
- En klient måste kunna få direkt kontakt med en annan klient (kallas då Peer). Detta därför att all filöverföring skall ske direkt mellan klienter (finns inga filer på servern). Kallas ofta peer2peer (P2P) nätverk.

Även servern startas med ett skript, se runserver.sh. Servern saknar användar-interface. Det enda som sker är att den skriver ut vad som händer i terminalen (använd log, se utdelad kod).

```
Trying to start server with registryport 6666 and s
erver port 7777
Server stub bound in registry at port 6666
Server running at localhost 7777
Feb 5, 2012 10:15:09 AM edu.gu.hajo.mbserver.core.S
erver register
INFO: register User[Fia]
Feb 5, 2012 10:15:09 AM edu.gu.hajo.mbserver.core.S
erver updateUsers
INFO: Updating users [Fia]
Feb 5, 2012 10:15:11 AM edu.gu.hajo.mbserver.core.S
erver register
INFO: register User[Pelle]
Feb 5, 2012 10:15:11 AM edu.gu.hajo.mbserver.core.S
erver updateUsers
INFO: Updating users [Pelle, Fia]
Feb 5, 2012 10:16:20 AM edu.gu.hajo.mbserver.core.S
erver updateUsers
INFO: Updating users [Fia]
Feb 5, 2012 10:16:20 AM edu.gu.hajo.mbserver.core.S
erver unRegister
INFO: unregister User[Pelle]
```

5 Domänmodell

Modellen är enkel (en Chat är associerad med 0 eller n andra Chat-objekt).



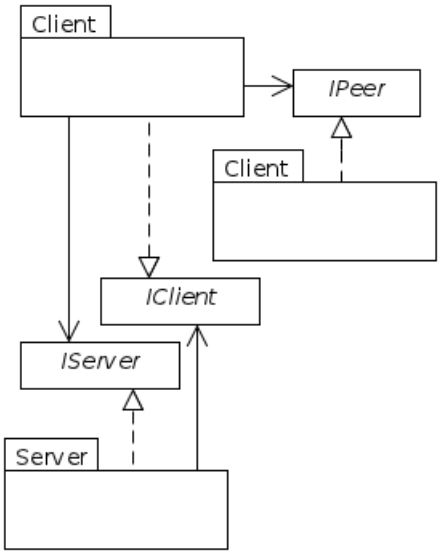
6 Övergripande process

Se vidare nedan.

1. Implementera preliminära versioner av några metoder i servers interface. Testa
2. Definiera klientens olika gränsnitt. Vad behöver servern kunna anropa? Vad behöver GUI:et? Avvakta med IPeer.
3. Implementera en metod i taget rakt igenom hela applikationen.

7 Systemarkitektur

Eftersom vi skriver flera samverkande applikationer kallar vi detta en arkitektur (mer övergripande än design).



Server implementerar gränssnittet IServer som används av Client. Client implementerar IClient som används av Servern. IPeer används mellan klienter vid filnedladdning. Samtliga gränssnitt "extends Remote" eftersom metodanropen sker med RMI, se utdelad kod.

VARNING: Alla metoder som skall användas remote måste ha "extends RemoteException". Alla objekt som skall skickas över nätet måste implementera Serializable! Annars undantag!

Uppgift 1 Det finns två Eclipse project att ladda ner, ett för klient och ett för server. Hämta dessa och importera till Eclipse. Source foldern "common" i serverprojektet innehåller klasser som är gemensamma för projektet. Man kan länka denna folder till klientprojektet (Build Path > Link Source). Både applikationerna är körbara men fungerar naturligtvis inte. Testa att starta server och klienter m.h.a. skripten.

Uppgift 2 En metod, ping, är implementerad i servern. Det finns en test i klienten som anropar metoden. Starta servern och kör testet (JUnit inifrån Eclipse).

Det finns noteringar TODO i koden där saker behöver åtgätdas.

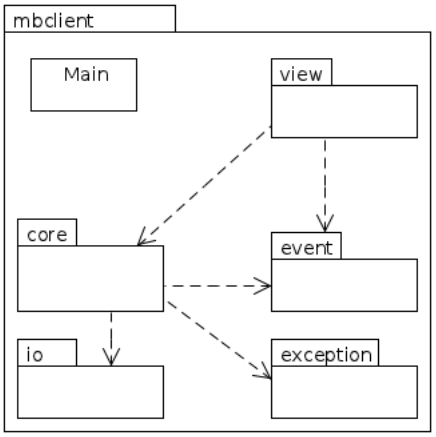
8 Designmodell för Servern

Det mesta av modellen ligger i common katalogen. Själva servern består bara av en enda klass. Dessutom finns en Main-klass för att starta server (den är färdig).

Uppgift 1 Vilka metoder behövs i serverns interface. Vad vill klienten anropa? Implementera och testa (några).

9 Designmodell för Klienten

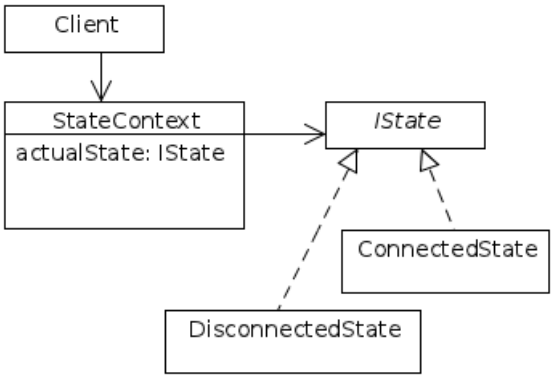
Generellt samma som i föregående lab (förenklad MVC). En övergripande bild på paketnivå.



- Main innehåller metoden main.
- view är hela GUI:et
- core är en del av modellen.
- event innehåller en EvenBus som används för observermönstret (isf PropertyChangeListener m.fl.)
- io hantrera läsning och skrivning av filer.
- exception innehåller en del klasser för undantag.

TIPS Låt EventBus finnas i Client-klassen och låt interfacet som GUI:et skall använda ha en metod getBus(). GUI-klasserna måste kunna registrera sig som observatörer.

Klienten kan uppenbart vara i två olika tillstånd, uppkopplad och nedkopplad. Utifrån detta skall klienten använda designmönstret State. Klienten byter alltså tillstånd genom att byta objekt. Klassen StateContext håller reda på de två tillstånden (objekten) samt det aktuella tillståndet. RMI hanteringen läggs i klasserna Connected och DisconnectedState (klienten startar alltid i nedkopplat läge).



Uppgift 1 Vilka metoder skall finnas i IState (vad behöver klienten? Jmf IServer). Implementera state-mönstret och testa (utan nätverksanrop). Om en metod är meningslös i något tillstånd låter man den bara vara tom (såvida inte ett undantag skall kastas).

Uppgift 2 Skapa ett klient-interface som skall användas av GUI:er, metoder? Lägg till någon.

Uppgift 3 Vilka metoder skall finnas i IClient (interfacet server använder)? Lägg till några (någon) metod.

Uppgift 4 Implementera en metod i taget hela väger från klientens GUI till server och tillbaks till klienten.

10 Felhantering

Felhanteringen är ganska komplex. Många olika undantag kan uppstå.

1. Klienten försöker ensluta sig till en server med ett namn som redan är upptaget.
2. Klienten kraschar vad gör servern?
3. Klient försöker ansluta till server som inte är igång.
4. Klient skickar null-parameter till servermetod?
5. Server returnerar null, skall vi tillåta sådant? Undantag?
6. Server kraschar, vad gör klienten?

Uppgift_1 RMI stödjer remote-exceptions d.v.s. felen skickas från server till klient (kan använda throw new RemoteException(...)). Innebär att felhanteringen främst sker på klientsidan. När några delar fungerar försök se över felhanteringen, försök vara konsekevent. Försök hitta en eller ett par abstraktionsnivåer där felen hanteras. Skilj på felhantering som sker i modellen (åtgärda fel om möjligt) och meddelanden till användaren, ren info (sker m.h.a. eventbus och dialogrutor). Undvik try..catch i GUI:et.

TIPS Låte en tråd på servern rensa ut "bottappade" klienter med jämna mellanrum.

11 Parallellism

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 klientsiden 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 (Swing worker för filnerladdning).

Uppgift 1 Se över allt som har med trådar att göra.

12 Filöverföring

Uppgift 1 Bestäm vilka metoder som behövs i IPeer. och implementera filöverföringen. Mycket finns färdigt, se t.ex. MBFile, det gäller att koppla ihop det hela.

13 Redovisning

Som tidigare labbar.

Inlämningsdatum Se kurssida.