Inlämningsuppgift 2 (P2) – Nätverkskommunikation

17/2 Frågestund kl. 15:15
24/2 Inlämning kl. 14:00
25-27/2 Kamratgranskning
28/2 + 29/2 Redovisning

1 Inledning

Den här inlämningsuppgiften ska bidra till en grundläggande förståelse för:

- Användning av trådar
- Användning av strömmar
- Nätverkskommunikation via TCP/IP
- Implementering av Callback och PropertyChangeListener

1.1 Filer som bifogas

TestP2Input.java, MainP2.java, Sound.java, mp3plugin.jar

1.2 Redovisning

Din lösning av **uppgiften lämnas in via Canvas senast kl 14:00 den 24/2** (det är tillåtet att lämna in tidigare). Inlämningen ska innehålla samtliga klasser som används i lösningen. Klasserna *MessageServer* och *MessageClient* ska vara javadoc-kommenterade och javadoc ska vara genererad. Javadoc filerna ska vara sparade i katalogen ”docs” i projektet.

Vid **redovisningen den 28/2 och 29/2** kommer din lösning att köras med programmet MainP2. Kontrollera därför noga att din lösning fungerar innan inlämningen. Se till att bifoga en tydlig instruktion så att granskaren kan exekvera din lösning.

Projektet och Zip-filen ska du ge namnet AAABBB_P2 där AAA är de tre första bokstäverna i ditt efternamn och BBB är de tre första bokstäverna i ditt förnamn. Använd endast tecknen a-z när du namnger filen.

- Om Rolf Axelsson ska lämna in sina lösningar ska filen heta AxeRol_P2.zip.
- Om Örjan Märla ska lämna in sina lösningar ska filen heta MarOrj_P2.zip.
- Är ditt förnamn eller efternamn kortare än tre bokstäver så ta med de bokstäver som är i namnet: Janet Ek lämnar in filen EkJan_P2.zip

1.3 Granskning

Senast kl 10.00 den 25/2 kommer en kamrats lösning finnas i din inlämning på Canvas. Din uppgift är att granska kamratens lösningar på uppgifterna avseende:

- funktion – hur väl uppfyller lösningen kraven i uppgiften? Fungerar klasserna på avsett sätt?
- kan du tänka dig något alternativt sätt att lösa uppgiften?
- javadoc-kommentarer – är klasserna kommenterade enligt instruktion? Och är kommentarerna vettiga? Är javadoc-dokument genererade? Fungerar länkar?

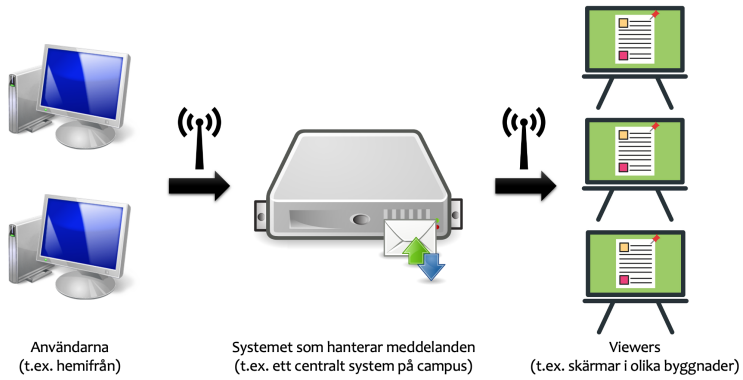
Resultatet av din granskning, 1-2 A4-sidor, ska du **lämna in via Canvas senast den 27/2**.

2 Beskrivning av uppgiften

P2 (inlämningsuppgift 2) är fortsättning på din lösning av P1 (inlämningsuppgift 1). Därför kräver P2 att P1 fungerar på avsett sätt. Klasserna i P1 återanvänds i P2, kanske med ett mindre tillägg i någon klass.

Nu ska systemet som kan visa meddelanden på olika viewers ändras så att kommunikationen sker via nätverk. Det innebär att användarna kan koppla upp sig mot systemet för att ladda upp sina meddelanden (*MessageProducer*) och att även skärmarna (*viewers*) får meddelanden med hjälp av nätverkskommunikation.

Med hänsyn till scenariot från första inlämningsuppgiften, ett system där användare kan ladda upp meddelanden som sedan visas på olika skärmar på universitetet, betyder det att användarna kan skicka meddelanden hemifrån till ett centralt system på campus som hanterar meddelanden. Alla skärmar, som kan befinna sig i olika byggnader (t.ex. Niagara, Orkanen, …), är uppkopplade till det centrala systemet via internet och tar emot meddelanden som ska visas.



Figuren nedan visar P1 i gröna färger. Dock inte hela systemet utan endast start (ljusgrön) och slut (mörkgrön) på P1. I P2 ska datakommunikation tillföras i systemet:

Röd färg (vänster sida):

1: En MessageProducerClient (någonstans i världen) ska kunna koppla upp mot en MessageProducerServer. Efter att klienten kopplat upp ska en MessageProducer-implementering
överföras till servern.

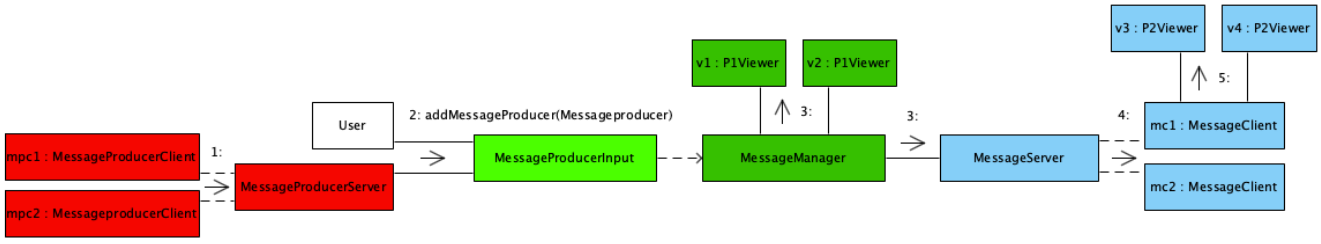
2: Servern anropar metoden addMessageProducer i MessageProducerInput.

Blå färg (höger sida):

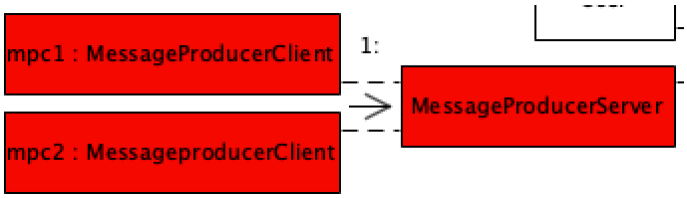
3: En MessageServer erhåller Message-objekt på samma sätt som P1Viewer erhåller Message-objekt, dvs genom Callback eller PropertyChangeListener.

4: MessageServern överför Message-objekten till uppkopplade MessageClients.

5: Dessa Message-objekt ska visas i P2Viewer.



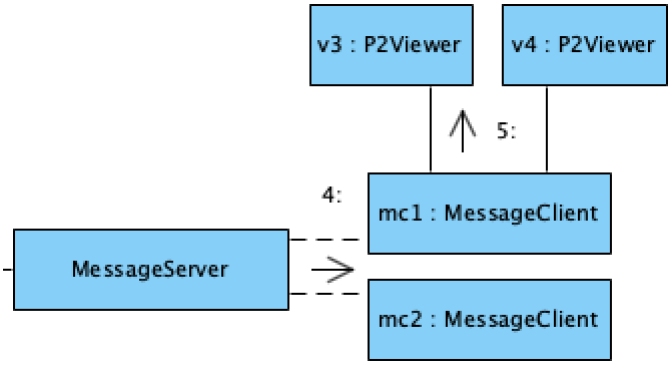
2.1 Design av röd sida



- ***MessageProducerClient*** måste känna till ip-adress och uppkopplingsport till MessageProducerServer. Klienten utgörs troligen av ett antal klasser (kan vara inre klasser). Klienten ska koppla upp mot server, överföra en MessageProducer-implementering och sedan koppla ner förbindelsen.
- ***MessageProducerServer*** måste veta på vilken port den ska lyssna för att hantera uppkopplingar. Servern måste också ha en referens till MessageProducerInput. Servern utgörs troligen av ett antal klasser (kan vara inre klasser). Servern ska vara en *iterativ server* som använder *en tråd*. Servern ska alltså hantera en klient i taget. Kommunikationen mellan klient och server sker med objektströmmar, dvs ObjectInputStream respektive ObjectOutputStream ska användas.
- ***ArrayProducer*** är lämplig att använda för att överföra en MessageProducer-implementering. Dock måste klassen modifieras så den fungerar i strömmar.

  Programmet ***TestP2Input*** ska du använda för test av vänster sida. Resultatet ska visa sig i *P1Viewers* då du använder programmet.

2.2 Design av blå sida



- ***MessageServer*** ska vara en *flertrådad server* där varje uppkopplad klient ska hanteras av en tråd. ***MessageManager*** ska överföra *Message*-implementeringar till ***MessageServer*** på samma sätt som *Message*-implementeringarna överförs till **P1*Viewer***-objekten i P1.
- ***MessageClient*** utgörs troligen av ett antal klasser (kan vara inre klasser). ***MessageServer*** och ***MessageClient*** ska kommunicera via objektströmmar (*ObjectInputStream* resp *ObjectOutputStream*). Det kan vara en god idé att låta klienten och servern vara uppkopplade mot varandra under hela exekveringen.
- ***P2Viewer*** liknar *P1Viewer* från P1. ***MessageClient*** ska överföra *Message*-objekt till ***P2Viewer***-objekten. Om överföringen av *Icon*-objekt till *P1Viewer* sker genom *Callback* så ska överföringen till *P2Viewer* ske genom *PropertyChangeListener* (använd också gärna klassen *PropertyChangeSupport*) och tvärt om ifall *P1Viewer* redan använder *PropertyChangeListener*. Om så är fallet ska *P2Viewer* erhålla *Icon*-objekt genom *Callback*.

För att testa hela systemet ska du använda MainP2. MainP2 körs också under redovisningen.

Det ska alltså fungera att exekvera användarens klient (*MessageProducerClient*), det centrala systemet som hanterar meddelanden och viewerna (*MessageClient*) på tre olika datorer. För redovisningen räcker det dock att exekvera alla tre delar på samma dator men gärna som tre separata program.

3 Extrauppgift (frivilligt!)

Det vore trevligt om ett Message även innehåller en ljudfil vilken spelas upp då bild och text visas.

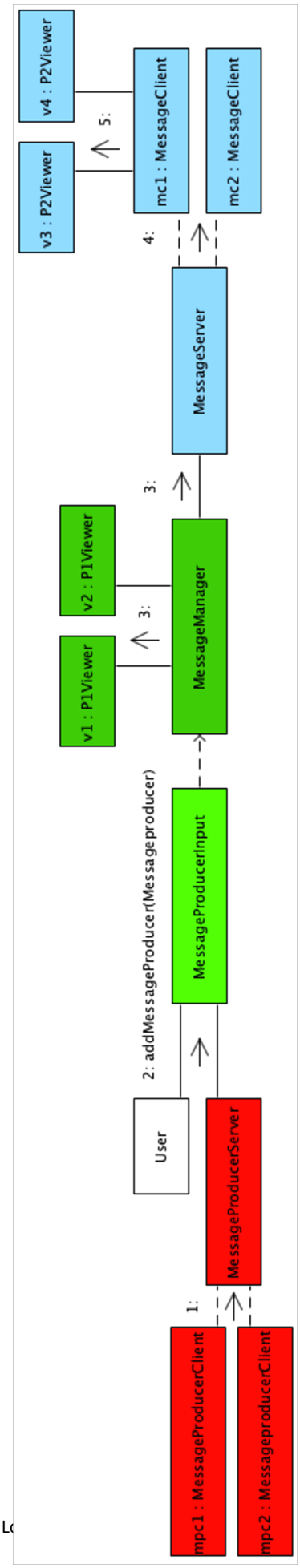
Om du vill ordna detta så:

Med uppgiften kommer ***Sound***-klassen vilken innehåller metoden ***getSound(byte[] byteSound)***. Så om en mp3-fil lästs in till en byte-array och överförts till en Viewer så kan ljudet spelas upp (om mp3-formatet stödjs av mp3plugin.jar).

Det krävs ett par ting:

En klass vilken ärver Message och som har en byte-array som instansvariabel. Vettigt är om denna subklass tar namnet på en ljudfil som input till konstruktorn och läser in filen i en byte-array. Till god hjälp kan ByteArrayOutputStream vara.

En Viewer vilken kan spela upp ljud om Message-objektet är en subklass som håller en bytearray med ljudligt innehåll. P2SoundViewer kan t.ex. ärva P2Viewer.


mpc1 : MessageProducerClient
mpc2 : MessageproducerClient
1:
MessageProducerServer
User
2: addMessageProducer(Messageproducer)
MessageProducerInput
MessageManager
3:
v1 : P1Viewer
v2 : P1Viewer
3:
MessageServer
4:
mc1 : MessageClient
mc2 : MessageClient
5:
v3 : P2Viewer
v4 : P2Viewer