Inlämningsuppgift 2, P2 – Nätverkskommunikation

1 Inledning

Den här inlämningsuppgiften ska bidra till en grundläggande förståelse för:

- Användning av trådar
- Användning av strömmar
- Nätverkskommunikation via TCP/IP
- Implementering av Observer eller Callback

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 09.00 onsdagen den 6/3 (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.

Vid redovisningen måndagen den 7/3 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.

Zip-filen ska du ge namnet AAABBBP2.zip 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 AxeRolP2.zip.
- Om Örjan Märla ska lämna in sina lösningar ska filen heta MarOrjP2.zip.
- Är ditt förnamn eller efternamn kortare än tre bokstäver så ta med de bokstäver som är i namnet: Janet Ek lämnar in filen EkJanP2.zip

1.3 Granskning

Ca kl 14.00 den 6/3 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? Fumger länkar?

Resultatet av din granskning, 1-2 A4-sidor, ska du lämna in via Canvas senast 13.00 den 7/3.

2 Beskrivning av uppgiften

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

Figuren nedan visar P1 i gröna färger. Dock inte hela systemet utan endast start (ljus grön) och slut (mörk grön) 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 .

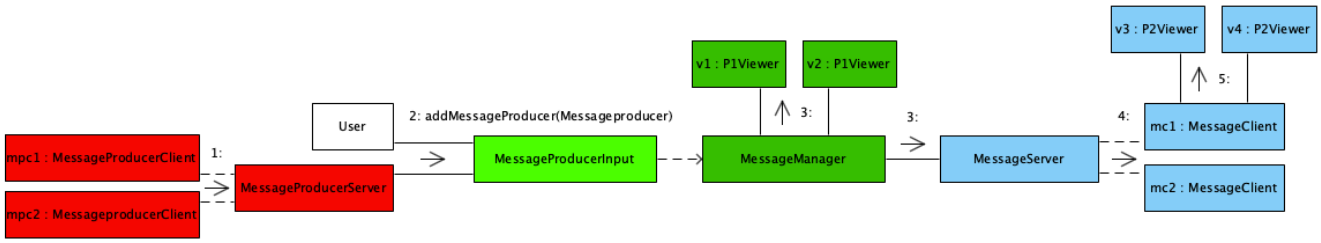
2: Servern anropar metoden addMessageProducer i MessageProducerInput.

Blå färg (höger sida):

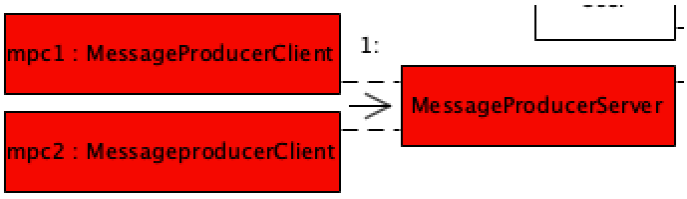
3: En MessageServer erhåller Message-objekt på samma sätt som P1Viewer erhåller Message-objekt, dvs genom Observer eller Callback.

4: MessageServern överför Message-objekten till uppkopplade MessageClients.

5: Dessa Message-objekt ska visas i P2Viewer.



2.1 Design av röd sida



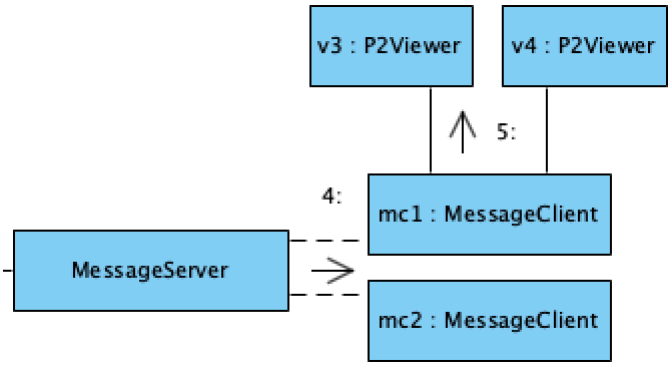
- ***MessageProducerClient*** måste känna till ip-adress och uppkopplingsport till MessageProducerServer. Klienten utgörs troligen av ett antal klasser (kan vara inre klasser). Klienten ska koppla upp mot server, överföra en MessageProducer-implementering och sedan koppla ner förbindelsen.

- ***MessageProducerServer*** måste veta på vilken port den ska lyssna 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 hanterar 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* (eget system för notifiering av lyssnare) så ska överföringen till *P2Viewer* ske genom designmönstret *Observer* (klassen *Observable* och interfacet *Observer* används). Och tvärt om ifall *P1Viewer* använder designmönstret *Observer*. Om så är fallet ska *P2Viewer* erhålla *Icon*-objekt genom *Callback*.

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

3 Extrauppgift

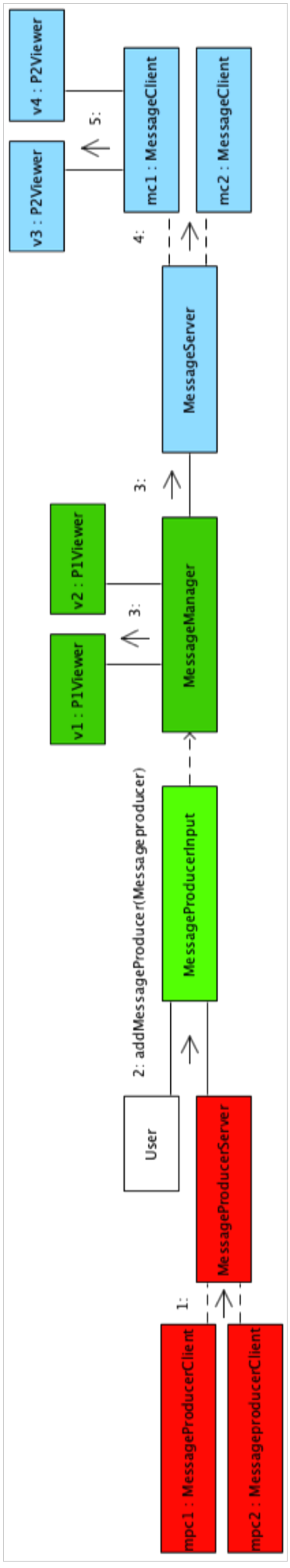
Det vore trevligt om ett Message även innehåller en ljudfil vilken spelas upp då bild och text visas. Om du vill ordna detta så:

Med uppgiften kommer en ny version av ***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
v1 : P1Viewer
v2 : P1Viewer
3:
MessageManager
3:
MessageServer
4:
v3 : P2Viewer
v4 : P2Viewer
5:
mc1 : MessageClient
mc2 : MessageClient