# Laboration – wild chat

*Denna laboration löses individuellt.*
*Det är ej tillåtet att kopiera en annans students lösning för aktuellt projekt.*
*Du kan bli ombed att redovisa din lösning muntligen.*

## Innehållsförteckning

## Bakgrund

OSC.org har påbörjat utvecklingen emot en nygammal-chat. Idén är att locka in den äldre generationen som känner sig ifrån sprungna med alla chatlösningar som presenteras i webbläsaren och saknar den gamla tiden (IRC, MSN-messanger med flera).

Därför utvecklas för närvarande en desktopchat i Java. Arbetet är i full gång och en prototyp finns på plats. Tyvärr har nu samtliga utvecklare av prototypen sagt upp sig.

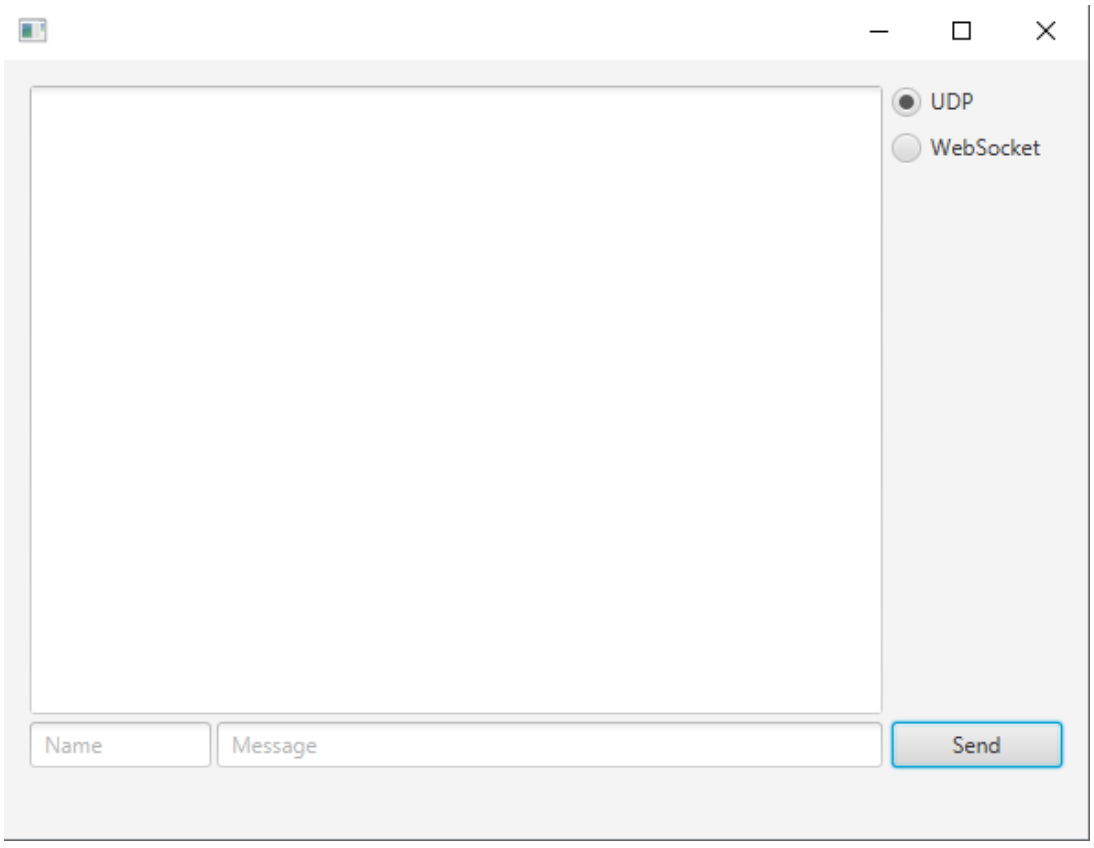
Det är här du kommer in i bilden, ditt uppdrag är att kliva in och se till att tänkt funktionalitet i applikationen fungerar. Självklart skall du då även se till att nyttja dina kunskaper inom designmönster för att skapa en robust lösning.

## Projektet

Kod att utgå ifrån hittas i anslutning till denna fil.

## Att göra

När du startar applikationen möts du av följande:




I chattens högra hörn finns det möjlighet att välja kommunikationsprotokoll, observera att för närvarande så händer inget bakom kulisserna när du ändrar radio buttons. Din uppgift är alltså att se till att det är möjligt att ändra kommunikationsprotokoll i applikationen.

*Nedan så följer genomgång av vad som skall implementeras i applikationen. Genomgången är uppdelad i delar (A, B och C (C endast för dig som siktar på VG)). Observera dock att dessa delar är starkt sammankopplade med varandra och även om det kan vara smidigt att tackla en av dem åt gången bör du först förstå helheten.* ***Alltså så är det mycket möjligt att din lösning för A behöver editeras när du tar dig an B alternativt. De presenteras endast som steg på grund av enkelheten att få tolka vad som skall lösas.***

## Deluppgift A (G)

Kod finns för att kommunicera antingen via UDP i (klassen **UDPChatCommunicator**) eller via Web Socket (klassen **WebSocketCommunicator**)**.** Men ingen logik finns för att skifta vilken som används när användaren av chatten ändrar val i applikationen.

Det är möjligt för dig att ändra vilken kommunikatör som används i koden genom att ändra vilken rad som är bort kommenterad (se rad 33–38 i MainWindowController.java).

```
//---------- Deluppgift A test:
//Change what line is commented to change communicator:
//private WebSocketCommunicator _communicator = new WebSocketCommunicator(this);
private UDPChatCommunicator _communicator = new UDPChatCommunicator(this);
//----------
```

Uppdraget i deluppgift A är att implementera så det i applikationen är möjligt att dynamiskt ändra vilken kommunikatör som används genom att klicka i radioknapparna.

**Din implementation skall innehålla minst ett vedertaget designmönster** *(som då inte är det som nämns i del B).*

## Deluppgift B (G)

Förnärvarande föreligger det en stark koppling emellan kontrollerklassen (MainWindowController) och kommunikationsklasserna (UDPChat/WebsocketCommunicator). Kommunikationsklasserna är således låsta till att endast kunna kommunicera med MainWindowController klassen och vice versa. Detta är något vi önskar luckra upp genom att implementera en Observer pattern lösning

Ditt uppdrag i deluppgift B är alltså att se till att en observer mönsterlösning finns med i applikationen i avseendet ”kommunikation emellan klasser”.

**(*OBS Javas native observer lösning får ej nyttjas.*)**.

## Deluppgift C (VG)

Programmet behöver utökas med loggnings funktionalitet. I denna första version bör samtliga meddelanden och fel i programmet loggas till System.out. Men tanken är att i framtiden kunna ändra hur loggningen sker (till exempel logga till olika typer av filer / databaser).

Ditt uppdrag i deluppgift C är: Chatmeddelanden och fel (exceptions) loggas till System.out tillsammans med datum och tid. Tänk på att göra en lösning så det är enkelt att i framtiden välja hur loggning sker. Till din kodlösning skall även en .pdf lämnas in som besvarar samt motiverar:

- Hur din lösning gör det enkelt att i framtiden ändra loggningsmetod.

- Vilket eller vilka designmönster använde du?

- För en diskussion/jämförelse kring minst ett annat designmönster som skulle kunnat använts för att lösa uppgiften.

## Allmänt

Du är fri att ändra, lägga till och förbättra samtlig kod i projektet.

## Rapport

Utöver en fungerade applikation skall du även lämna in rapport.

I rapporten skall det framgå:

- Vilka mönster som har använts.
- Hur dessa har implementerats i applikationen (text samt illustrationer).
- Hur förhåller sig mönstren och arbetet du gjort i projektet till SOLID principerna (*lyft både styrkor och svagheter, samtliga principer behöver inte tas med om du inte har något intressant att nämna, men det är heller inte en ursäkt att ”hoppa över en stor mängd av dem”*).
- Innehållslängd och ”struktur” bör minst möta och helst överträffa uppgiftsbeskrivningen.

# Förberedelser

- Föreläsningar och övningar.
- Tidigare examinationer.
- Kursbok.
- Övriga källor.

# Inlämning

Du laddar upp en .zip innehållandes:

1. Koden för en fungerande lösning av applikationen med efterfrågad funktionalitet.
2. En .pdf (rapport) med efterfrågat innehåll.

## Betyg

Inlämningen kan ge något av betygen komplettera (U)*, godkänd (G) eller väl godkänd (VG).

För att ha chans på betyget G skall: deluppgift A och B vara lösta.

För att ha chans på betyget VG skall: deluppgift A, B och C vara lösta.

**Om din inlämning inte lever upp till kraven för G har du möjlighet att komplettera uppgiften för att nå godkänd.*
*Om din inlämning ej lever upp till kraven för väl godkänd har du regel ej möjlighet att komplettera upp till väl godkänd. Men ifall din inlämning är nära VG men slarv eller enklare fel förekommer kan du ges möjlighet att komplettera till väl godkänd.*

## Betygskriterier

När vi betygsätter inlämningen tittar vi på:

### *KOD:*

- Funktionaliteten i applikationen är som förväntat.
- Koden är enkel att underhålla och bygga vidare på.
- Klasser, metoder, variabler och package har tydliga och informativa namn som följer Javas namnkonvention.
- Indentering och övrig kodstruktur är tydlig och konsekvent.
- Klasser och metoder är kommenterade enligt JavaDoc, du har även lagt till ditt namn som Author ifall du har rört klassen.
- Klasserna är indelade i lämpliga paket.
- SOLID principer eftersträvas.

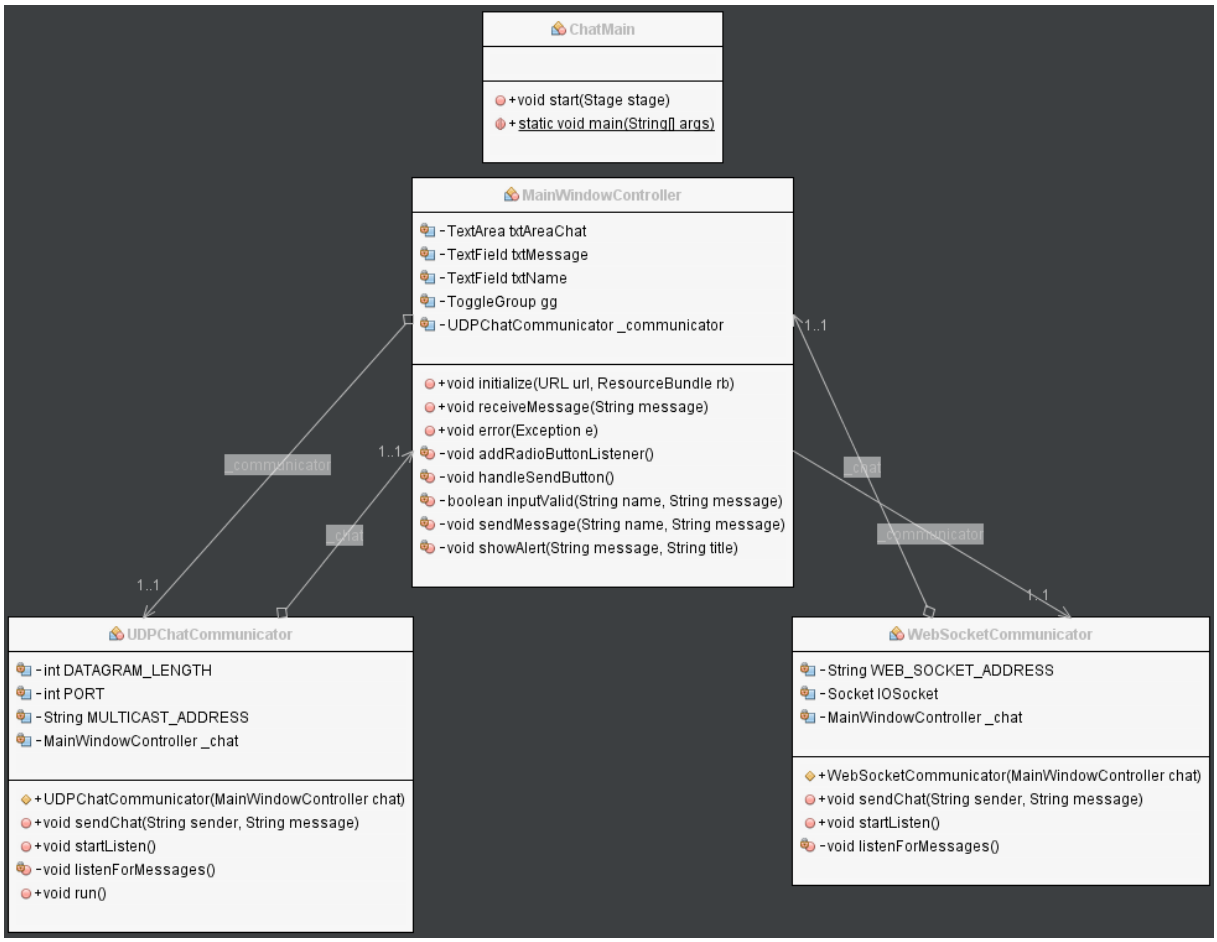
### *RAPPORT:*

- Efterfrågat innehåll finns.
- Tydlighet i skrift samt illustrationer.
- Stavning och grammatik.
- Rimliga motiveringar gällande SOLID principer.

### *För VG tittas även på:*

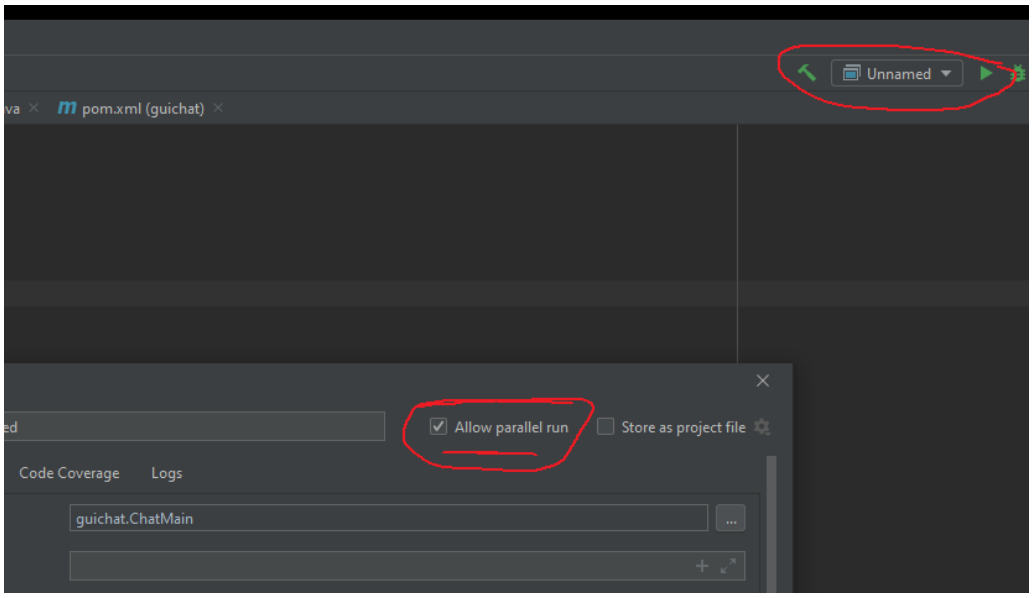
- Det är enkelt att lägga till eller ändra loggning.
- Hög kvalitet på lösningar för deluppgift A, B och C.

## UML nuvarande design




## Starta applikationen

Vid testning av chatten är det enklast att starta två chatfönster via din utvecklingsmiljö. Något som i regel kan göras via ”run” + ”Debug”. I Intellij behöver detta tillåtas i din run konfiguration:




När det är valt kan du sedan klicka ”run” två gånger.