Lab 4 : GUIs

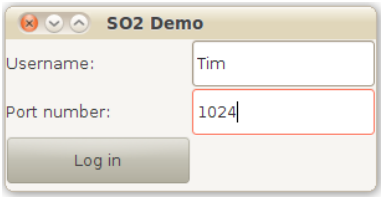
29 maart 2013

Tim Verbelen, Steven Van Canneyt, Hendrik Moens
(e-mail: {voornaam.achternaam}@intec.UGent.be)

1 Inleiding

In deze labsessie bouwen we een chattoepassing met een grafische front-end. Om de chatberichten tussen twee gebruikers te versturen, maken we gebruik van de genetwerkte `EventBroker` zoals we deze in de vorige labsessies gerealiseerd hebben. De grafische toepassing zal dus `Event`s genereren en ontvangen, die de chatboodschappen bevatten.
In deze labsessie wordt van de ontwikkelomgeving NetBeans gebruik gemaakt, omwille van de ingebouwde, Swing-compatibele GUI-builder (die het mogelijk maakt om met relatief weinig inspanning snel een grafische toepassing te bouwen). We veronderstellen dat bij aanvang van de labsessie de tutorial die via Minerva beschikbaar werd gesteld, doorgenomen werd (of dat de kennis erin vervat via een alternatieve weg verworven werd).

2 Login venster



Figuur 1: Login venster.

2.1 Opgave 1

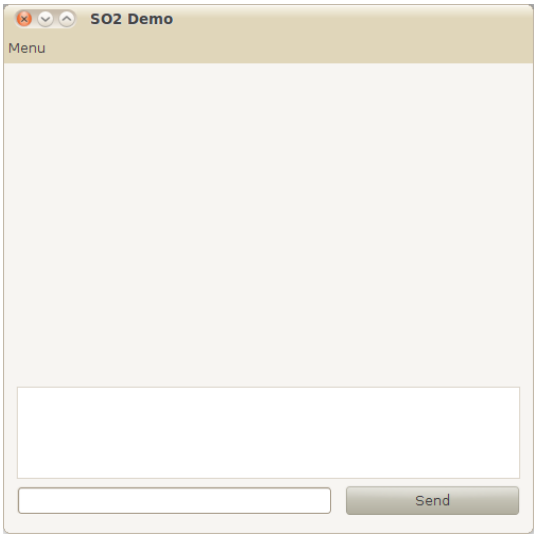
Maak een main klasse `main.Main`, en programmeer een loginvenster zoals Figuur 1 weergeeft. Hierbij wordt de naam van de gebruiker opgevraagd, samen met het poortnummer waarlangs de chattoepassing over het netwerk zal communiceren. Wanneer de “Log in”-knop aangeklikt wordt, wordt de invoer nagekeken op geldigheid. Is deze ongeldig, dan worden de tekstvelden gewist, en wacht de toepassing op een nieuwe klik op de knop “Log in”. Is de inhoud geldig, dan vervolgt de chattoepassing (zie Opgave 4). De invoer is ongeldig, indien

- de lengte van de usernaam 0 karakters bedraagt
- het poortnummer geen geheel getal voorstelt
- het poortnummer wel een geheel getal voorstelt, maar dit getal niet in het bereik $0..2^{16}-1$ ligt

Bouw dit loginvenster ZONDER gebruik te maken van de NetBeans GUI-builder. Kies voor een geschikte `LayoutManager` om dezelfde layout als in Figuur 1 te bekomen. Zorg er ook voor dat de GUI correct geïnitialiseerd wordt (= op de zogenaamde *event dispatching thread*).

3 Chat panel

Eens ingelogd ziet de gebruiker een venster zoals weergegeven in Figuur 2. Bovenaan is er een menubalk toegevoegd met één drop-down menu. Onderaan is er een chat panel, bestaande uit een venster dat de chat weergeeft, een tekst veld om zelf een bericht te typen en een knop om een bericht te versturen. In de volgende opgaven zullen we stapsgewijs hiernaartoe werken.

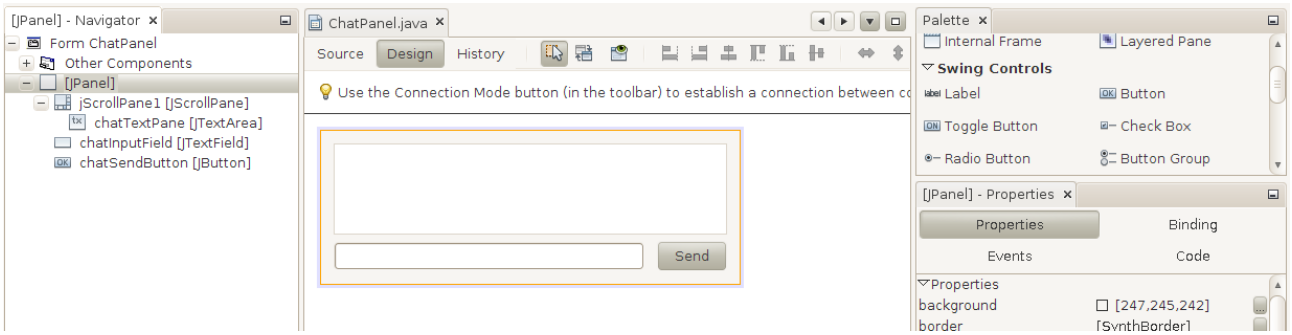


Figuur 2: Venster na inloggen, bestaande uit een menubalk en een chat panel.

3.1 Opgave 2

We starten met het ontwerpen van het chat panel, met behulp van de NetBeans GUI-builder. In de NetBeans IDE, kies "**File** >**New File**". Kies als categorie "**Swing GUI Forms**" en als type "**JPanel Form**" en klik "**Next**". Als klasse naam geef je `ChatPanel` op, en als package naam `chat`. Het bestand "ChatPanel.java" kan je bekijken in twee modi: design modus of source modus. De design modus laat je toe door middel van drag-and-drop GUI componenten op het panel te slepen, zoals weergegeven in Figuur 3. Dit maakt het makkelijk om complexe layouts te creëren zonder één regel code te moeten schrijven. Daarnaast kan je ook alle properties van de GUI componenten aanpassen in het Properties venster. De source modus geeft je de traditionele broncode editor, maar een heleboel code kan je niet wijzigen: dit is de broncode die de GUI-builder voor jou gegenereerd heeft.

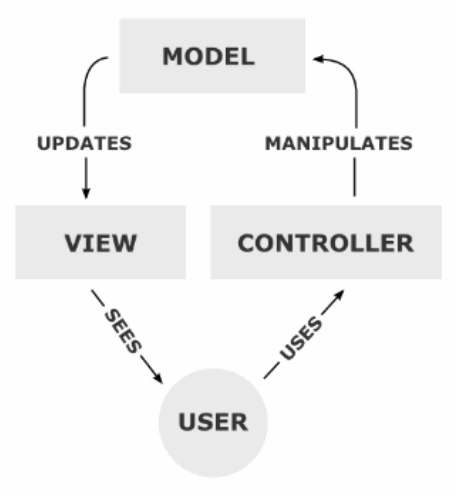
Ontwerp het `ChatPanel` zoals weergegeven in Figuur 3. Voeg ook een callback functie toe die wordt opgeroepen wanneer op de knop gedrukt wordt. Dit kan je makkelijk doen door in design modus te dubbel klikken op de knop. Kijk naar de genereerde broncode: begrijp je hoe en waar deze callback wordt opgeroepen? Laat deze methode voorlopig leeg, deze wordt ingevuld in de volgende opgave.



Figuur 3: De NetBeans GUI-builder laat je toe om door middel van drag-and-drop GUI componenten op het panel te slepen.

4 Model-View-Controller

Het Model-View-Controller patroon is een veelgebruikt patroon met als doel de GUI (= View) te scheiden van de data die deze weergeeft (= Model), en de achterliggende logica (= Controller). De werking ervan wordt weergegeven in Figuur 4. De gebruiker kijkt naar de View, en triggert acties van de Controller. De Controller past indien nodig het Model aan, en het Model notificeert de View dat deze gewijzigd is, en dat de GUI up to date moet worden gebracht.



Figuur 4: Model-view-controller

4.1 Opgave 3

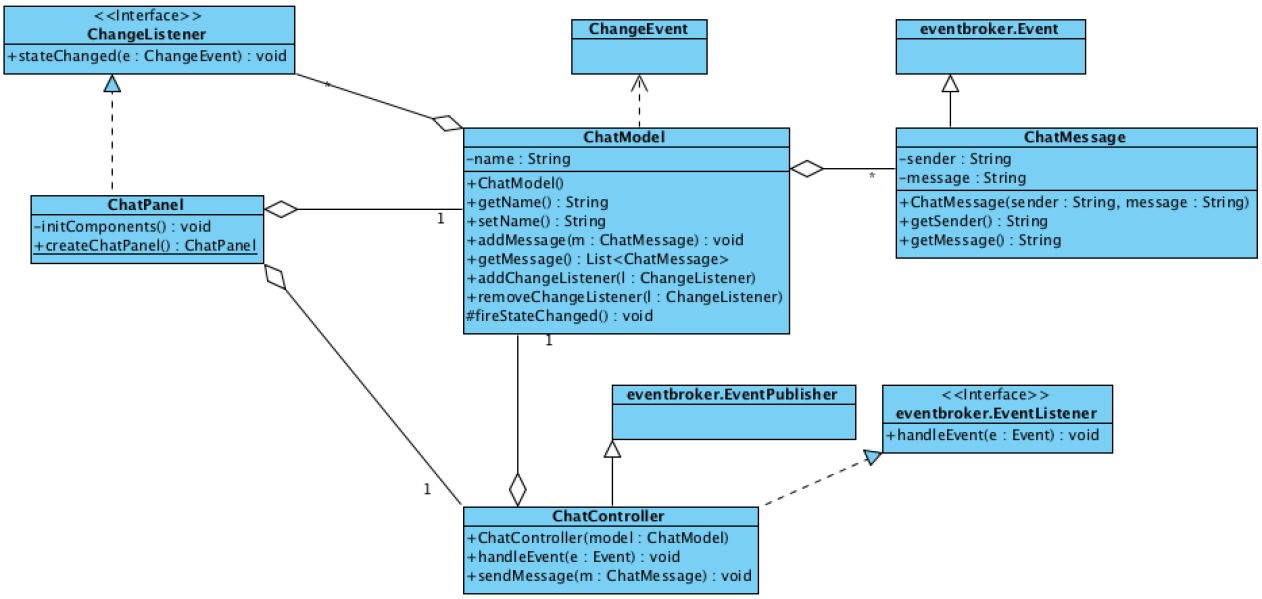
In deze opgave bouwen we de chattoepassing zelf via het MVC-patroon (figuur 4), waarbij de View-component het resultaat is van oefening 2. Het `chat.ChatModel` houdt dus alle relevante gegevens bij van de chatsessie, namelijk:

- de naam van de gebruiker die de chatsessie startte (voorzie een getter/setter-paar)
- de berichten van de chatsessie

Een bericht van de chatsessie wordt voorgesteld als een instantie van de klasse `chat.ChatMessage`. Een object van die klasse houdt de zender (= zijn naam) en de inhoud van het bericht bij. Aangezien chatberichten ook over het netwerk dienen verstuurd met behulp van de `EventBroker` uit vorige lab sessies, erft deze klasse over van de klasse `eventbroker.Event`.
Telkens de toestand van het `chat.ChatModel` wijzigt, wordt dit via het Observer-patroon doorgegeven aan mogelijke luisteraars. Hiervoor kan het gebruik maken van de Swing-types `ChangeEvent` en `ChangeListener`.
De `chat.Controller` implementeert de logica van de chattoepassing, en is verantwoordelijk voor het toevoegen van chatberichten aan `chat.ChatModel`. Daartoe luistert deze naar berichten die verstuurt worden vanuit de GUI enerzijds, en berichten die binnenkomen van andere gebruikers over het netwerk anderzijds.

- Wanneer op de “send” knop gedrukt wordt op de GUI, zal deze via de callback het bericht doorgeven aan `chat.Controller` met behulp van de `sendMessage()` methode. De `Controller` voegt dit bericht vervolgens toe aan het `ChatModel` en publiceert dit tevens bij de `EventBroker` als nieuw event, waardoor het over het netwerk verstuurd zal worden.
- De `Controller` luistert tevens naar `Event`s van `EventBroker`, en indien er een chatbericht binnenkomt wordt dit toegevoegd aan het `ChatModel`.

Onderstaand UML-klassendiagram (figuur 5) geeft een mogelijk ontwerp van de chattoepassing.



Figuur 5: Model-view-controller

Implementeer de klassen `chat.ChatMessage`, `chat.Controller` en `chat.ChatModel`. Voeg ook de factory methode `public static ChatPanel createChat(String name)` toe aan de klasse `chat.ChatChatPanel`, dewelke een `ChatPanel`, `Controller` en `ChatModel` aanmaakt en de juiste luisteraars configureert. Om het chat panel te testen dienen we dit te tonen na inloggen, wat we behandelen in Opgave 4.

Let er opnieuw op dat alle aanpassingen aan de GUI op de Swing *event dispatching thread* gebeuren, in het bijzonder bij `stateChanged()`.

5 Opgave 4

In deze opgave integreren we alle delen tot een werkende chat applicatie. Gebruik hiervoor de multithreaded event broker (package `eventbroker`) uit lab sessie 3, en het netwerk subsysteem (package `network`) uit lab sessie 2.

Wanneer de gebruiker succesvol inlogt, dienen de volgende acties te gebeuren:

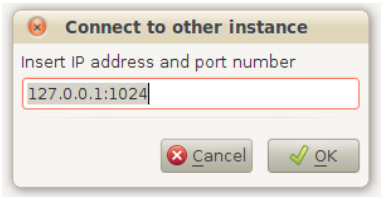
- Het frame wordt leeggemaakt.
- De `EventBroker` wordt gestart (zie lab sessie 3 voor de multithreaded `EventBroker`).
- Een nieuwe instantie van `Network` wordt aangemaakt, luisterend op de poort meegegeven bij het inloggen.
- Een `ChatPanel` (en bijhorende `Model` en `Controller`) wordt aangemaakt en onderaan het frame toegevoegd (zoals weergegeven in Figuur 2). Gebruik hievoor de daartoe geïmplementeerde factory methode uit Opgave 3.
- Tot slot moet het frame opnieuw hertekend worden.

Implementeer deze stappen na succesvol inloggen. Om te testen of het chatten werkt, connecteer je twee locale instanties met behulp van de `connect()` methode van de klasse `Network`. Gebruik hierbij voorlopig vast gecodeerde IP-adressen (localhost) en poortnummers. In de volgende opgave voorzien we een menubalk die toelaat dit via de GUI te doen.

Merk op dat in dit geval beide instanties van de applicatie zowel de rol van client als server kunnen opnemen. Wanneer een connectie is opgezet, moeten beiden tegelijkertijd berichten kunnen versturen en ontvangen.

6 Menus en dialogs

Het venster weergegeven in Figuur 2 heeft ook een menu bar, bestaande uit één drop-down menu “Menu”. Dit bevat één menu-item “Connect...” dat het onderstaande dialoogvenster toont (Figuur 6), hetgeen toelaat te connecteren naar een willekeurige andere client, gespecificeerd via IP-adres en poortnummer.



Figuur 6: Connect dialog.

6.1 Opgave 5

Voeg daartoe aan Aanklikken van dit menu-item laat een dialoogvenster verschijnen, waar de gebruiker een ip-adres en poortnummer kan opgeven, waarna een verbinding met een client op die locatie opgezet wordt.

Voeg een menu bar toe aan het frame, met een drop-down menu “Menu” dat het menu-item “Connect...” bevat. Het toevoegen van deze menu bar gebeurt als extra stap na succesvol inloggen, zoals behandeld in Opgave 4. Bij het klikken “Connect...”, wordt een dialoogvenster getoond zoals weergegeven in Figuur 6, waar de gebruiker een ip adres en poortnummer kan ingeven om mee te connecteren.

7 Tot slot

Upload je bestanden naar Indianio. We verwachten volgende bestanden:

- package `main`
  - `Main.java`: het hoofdprogramma dat je toepassing aanstuurt.
- package `chat`
  - `ChatMessage.java`
  - `ChatPanel.java`
  - `ChatPanel.form`
  - `ChatModel.java`
  - `ChatController.java`
- packages `eventbroker` en `network`

Mogelijke uitbreidingen:

- Geef de gebruiker foutboodschappen wanneer hij ongeldige parameters ingeeft bij het inloggen of connecteren.
- Zorg dat de gebruiker ook een bericht kan versturen bij het drukken op de “enter” toets.