A02 / CHAT FÜR SCHWERHÖRIGE

Implementierung des Decorator Patterns

20. NOVEMBER 2014

ABLEITINGER & ERCEG
4AHITT

Inhalt

1. Github

Github-Link: https://github.com/serceg-tgm/A02-Chat_fuer_Schwerhoerige

Github-Tag: ableitinger_erceg_a02_v1

2. Aufgabenstellung

Dies ist eine Aufgabe für 2 Personen.

Es wird ein einfaches Chat-Programm für "Schwerhörige" erstellt, mit dem Texte zwischen zwei Computern geschickt werden können.

Dabei soll jeder gesendete Text "geschrien" ankommen (d.h. ausschließlich in Großbuchstaben, lächelnd wird zu *lol*, Buchstaben werden verdoppelt, … - ihr dürft da kreativ sein)

Zusätzlich sollen "böse" Wörter ausgefiltert und durch "$%&*" ersetzt werden. Diese Funktionalität soll aber im Interface jederzeit aktiviert und deaktiviert werden können.

Verwendet wird dafür ausgiebig das Decorator-Pattern.

3. detaillierte Arbeitsaufteilung mit Aufwandabschätzung

Teilaufgabe	beteiligte Teammitglieder	benötigte Gesamtzeit
UML-Diagramm erstellen	Ableitinger, Erceg	60 Minuten (1 Stunde)
Implementierung der Funktionalitäten inkl. JavaDoc	Ableitinger, Erceg	120 Minuten (2 Stunden)
Implementierung der GUI inkl. JavaDoc	Ableitinger, Erceg	60 Minuten (1 Stunde)
Programmtestung mittels JUnit	Ableitinger, Erceg	120 Minuten (2 Stunden)
Protokoll schreiben	Ableitinger, Erceg	120 Minuten (2 Stunden)
Gesamt		**480 Minuten (8 Stunden)**

4. anschließende Endzeitaufteilung

4.1 Ableitinger

Teilaufgabe	Datum	Zeit
UML-Diagramm erstellen	13.11.2014	25 Minuten
Implementierung der Funktionalitäten inkl. JavaDoc	18.11.2014	85 Minuten
Protokoll schreiben	19.11.2014	20 Minuten
Implementierung der Funktionalitäten inkl. JavaDoc	19.11.2014	80 Minuten
Implementierung der GUI inkl. JavaDoc	20.11.2014	20 Minuten
Programmtestung mittels JUnit	20.11.2014	70 Minuten
Gesamt	*20.11.2014*	**300 Minuten (5 h)**

4.2 Erceg

Teilaufgabe	Datum	Zeit
UML-Diagramm erstellen	13.11.2014	25 Minuten
Protokoll schreiben	13.11.2014	35 Minuten
Implementierung der GUI inkl. JavaDoc	17.11.2014	70 Minuten
Implementierung der Funktionalitäten inkl. JavaDoc	17.11.2014	25 Minuten
Implementierung der Funktionalitäten inkl. JavaDoc	19.11.2014	50 Minuten
Programmtestung mittels JUnit	20.11.2014	70 Minuten
Protokoll schreiben	20.11.2014	55 Minuten
Gesamt	*20.11.2014*	**330 Minuten (5 h 30 min)**

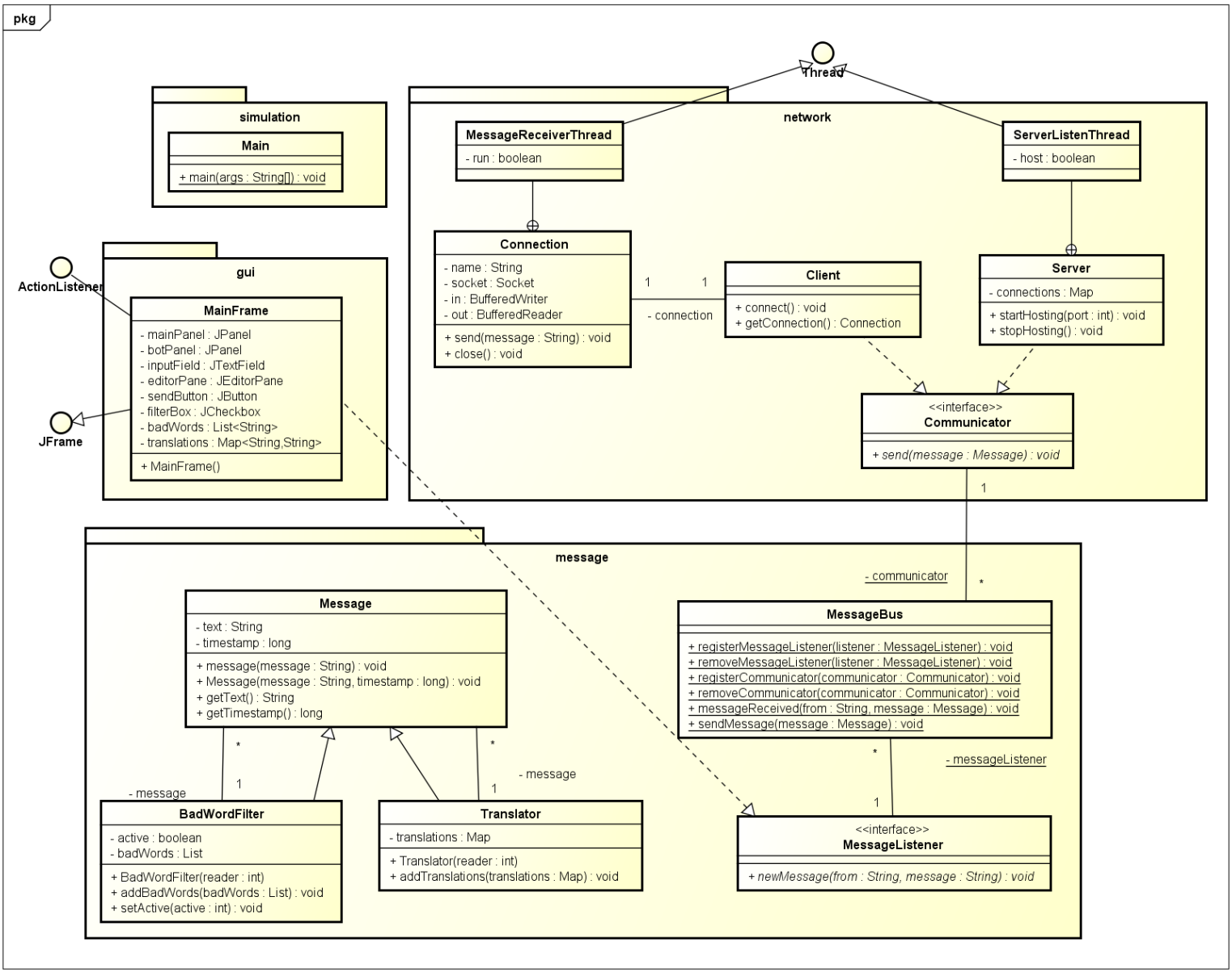
4.3 Gesamtsumme

Teilaufgabe	Datum	Zeit
UML-Diagramm erstellen	13.11.2014	50 Minuten
Implementierung der Funktionalitäten inkl. JavaDoc	19.11.2014	240 Minuten
Implementierung der GUI inkl. JavaDoc	20.11.2014	90 Minuten
Programmtestung mittels JUnit	20.11.2014	140 Minuten
Protokoll schreiben	20.11.2014	110 Minuten
Gesamt	*20.11.2014*	**630 Minuten (10 h 30 min)**

Für diese Aufgabe wurden insgesamt 10 ½ Stunden benötigt, geplant waren 2 ½ weniger. Dies lag am meisten daran, dass wir für die Implementierung der Funktionalitäten 2 Stunden länger brauchten als geplant.

5. Designüberlegung

5.1 Abbildung



5.2 Überlegungen zur Struktur

Wir haben uns vorgenommen, unser Programm in 4 Packages unterzuordnen:

1.) message

enthält die Message-Klasse mit all seinen Decoratorn (BadWordFilter und Translator). Außerdem befindet sich in dem Package das MessageListener-Interface, welches dazu verwendet wird, um sich bei dem MessageBus zu registrieren. Der MessageBus fungiert als zentrale Schnittstelle, da über ihn auch Nachrichten versendet werden.

2.) network

enthält die Klassen, welche für die Kommunikation zwischen den Chat-Clients notwendig sind. Beim Starten des Programms wird entweder ein Client- oder ein Server-Objekt instanziert, welches für die weitere Kommunikation, das Senden und Empfangen der Nachrichten, notwendig ist.

3.) gui

enthält die Klasse „MainFrame“, in der alle für die GUI notwendigen Elemente initialisiert werden und das Verhalten beim Drücken bestimmter Buttons (z.B. „Senden“) definiert wird.

4.) simulation

In diesem Package befindet sich die Main-Klasse, welche entweder den Client oder den Server startet.

6. Arbeitsdurchführung

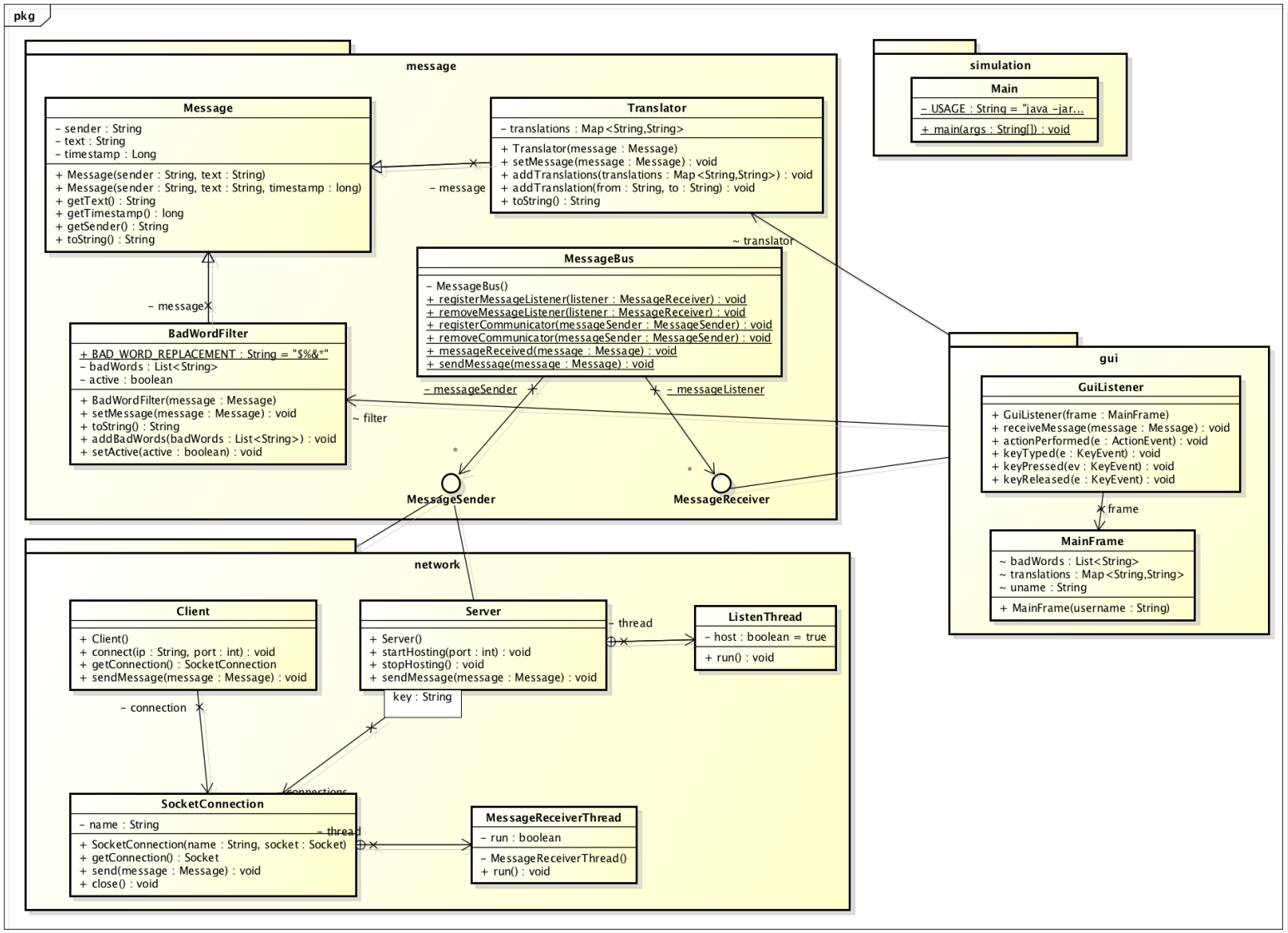
6.1 verworfene Überlegungen

Anfänglich wollten wir den BadWordFilter und den Translator als Reader-Decorator schreiben, welche die Input-Streams dekorieren. Dabei sind jedoch einige Probleme aufgetreten:

- Um Stream-Decorator effektiv nutzen zu können, müsste man jedes Zeichen einzeln behandeln. Da man aber beim BadWordFilter Wörter setzen muss, müsste man die bestimmten Zeichen zwischenspeichern, bearbeiten und dann erneut kopieren. Es wäre außerdem inperformanter, keinen Decorator zu verwenden.

- Sollten in späteren Anforderungen keine Zeichen (Strings) übertragen werden, sondern beispielsweise Objekte, würde das System nicht funktionieren.

6.2 veränderte Struktur



6.3 BadWordFilter und Translator

Im Folgenden sieht man die toString()-Methode des Translators. Diese ersetzt einfach jedes Wort (es findet also nur ganze Wörter und kein Teilvorkommen in der Nachricht) mit den vorher definierten Übersetzungen, also z.B. „lol“ zu „*ha*“. Am Ende wird einfach die Nachricht im upper case zurückgegeben.

```
public String toString() {

    String[] words = this.message.toString().split(" ");

    for(int i = 0; i < words.length; i++)
        for(Map.Entry<String, String> entry : translations.entrySet())
            if(words[i].equalsIgnoreCase(entry.getKey()))
                words[i] = entry.getValue();

    return String.join(" ", words).toUpperCase();
}
```

Im Folgenden sieht man die toString()-Methode des BadWordFilters. Diese ersetzt, nach einem ähnlichen Prinzip wie der Translator, einfach alle „bösen“ Wörter, welche vorher definiert werden müssen, mit der vorgegebenen Wortersetzung.

6.4 MessageListener in der SocketConnection-Klasse

Hier handelt es sich um eine Schleife, welche durch das Setzen des Flags „run“ beendet werden kann. Die Schleife ruft die „readObject()“-Methode des InputStreams auf und benachrichtigt den MessageBus, wenn eine neue Nachricht eingetroffen ist.

```
public void run() {

    try {

        while(run) {
            Object obj = in.readObject();
            if(obj instanceof String) {

                MessageBus.messageReceived(name, new Message((String) obj, 0));
            } else if(obj instanceof Message) {

                MessageBus.messageReceived(name, (Message) obj);
            } else {

                Main.LOG.error("Received illegal Object: " + obj.getClass());
            }
        }
    } catch(EOFException ex) {

        SocketConnection.this.close();
        throw new RuntimeException(ex);
    } catch (IOException | ClassNotFoundException ex) {

        throw new RuntimeException(ex);
    }
}
```

6.5 Main Methode

In der Main Methode wird, je nachdem welches Command Line-Argument angegeben wird, entweder ein Server oder ein Client gestartet. Der Client versucht außerdem ständig sich mit dem Server zu verbinden, wenn er keinen gefunden hat.

```
if(args.length == 0) {

    System.out.println(USAGE);
    return;
}

if(args[0].equalsIgnoreCase("server")) {

    if(args.length != 3) {

        System.out.println(USAGE);
        return;
    }

    Server server = new Server();
    server.startHosting(Integer.parseInt(args[1]));
    username = args[2];
} else if(args[0].equalsIgnoreCase("client")) {

    if(args.length != 4) {

        System.out.println(USAGE);
        return;
    }

    Client client = new Client();

    while(true) {

        try {

            client.connect(args[1], Integer.parseInt(args[2]));
            username = args[3];
            break;
        } catch(NumberFormatException ex) {

            System.out.println("Port not a number");
            System.out.println(USAGE);
            return;
        } catch(Exception ex) {

            LOG.warn("Couldn't connect to Server, retrying in 2 seconds");
            try { Thread.sleep(2000); } catch(InterruptedException ignore) { }
        }
    }
} else {

    System.out.println(USAGE);
    return;
}
```

7. Testbericht

Arbeitspaket	Erwartetes Ergebnis	Tatsächliches Ergebnis
BadWordFilter	„Bad Words“ werden bei der Nachricht nicht angezeigt bzw. „zensiert“	Stimmt mit erwartetem Ergebnis überein
Translator	Bestimmte Wörter werden bei der Nachricht mit einem anderen Wort ersetzt	Stimmt mit erwartetem Ergebnis überein
Senden der Nachrichten	Bestimmte Nachricht wird von einem Benutzer gesendet und beim Abschicken auf seinem System angezeigt	Stimmt mit erwartetem Ergebnis überein
Erhalten der Nachrichten	Nachricht kommt bei gewünschtem Benutzer an	Stimmt mit erwartetem Ergebnis überein
Filter-Funktion ein- bzw. ausschalten	Aktivierung bzw. Deaktivierung des BadWordFilters mittels einer Checkbox möglich	Stimmt mit erwartetem Ergebnis überein

8. Lessons learned

- gelernt, wann es sinnvoll ist, ein Decorator-Pattern anzuwenden
- Stream-Decorator sind eher nicht einsetzen, wenn man ganze Wörter behandeln muss und nicht einzelne Zeichen
- Ablauf beim Verbindungsaufbau zwischen Server und Client genauer betrachtet und Kenntnisse daraus gewonnen