

Chat - App

Einführung / Demo – Chat App

Features:

- App konfigurieren
- User Discovery
- Chat mit anderen Usern

Einführung / Demo

Starten der App mit Docker:

```
docker build -t aswe-chat-app .
```

```
docker run -it --network="host" aswe-chat-app
```

Alternativ mit Gradle:

Windows

```
gradle build
```

```
cd execution
```

```
execute-shadow-jar-develop.bat
```

Unix

```
chmod +x gradlew
```

```
./gradlew build
```

```
cd execution
```

```
chmod +x execute-shadow-jar-develop.sh
```

```
execute-shadow-jar-develop.sh
```

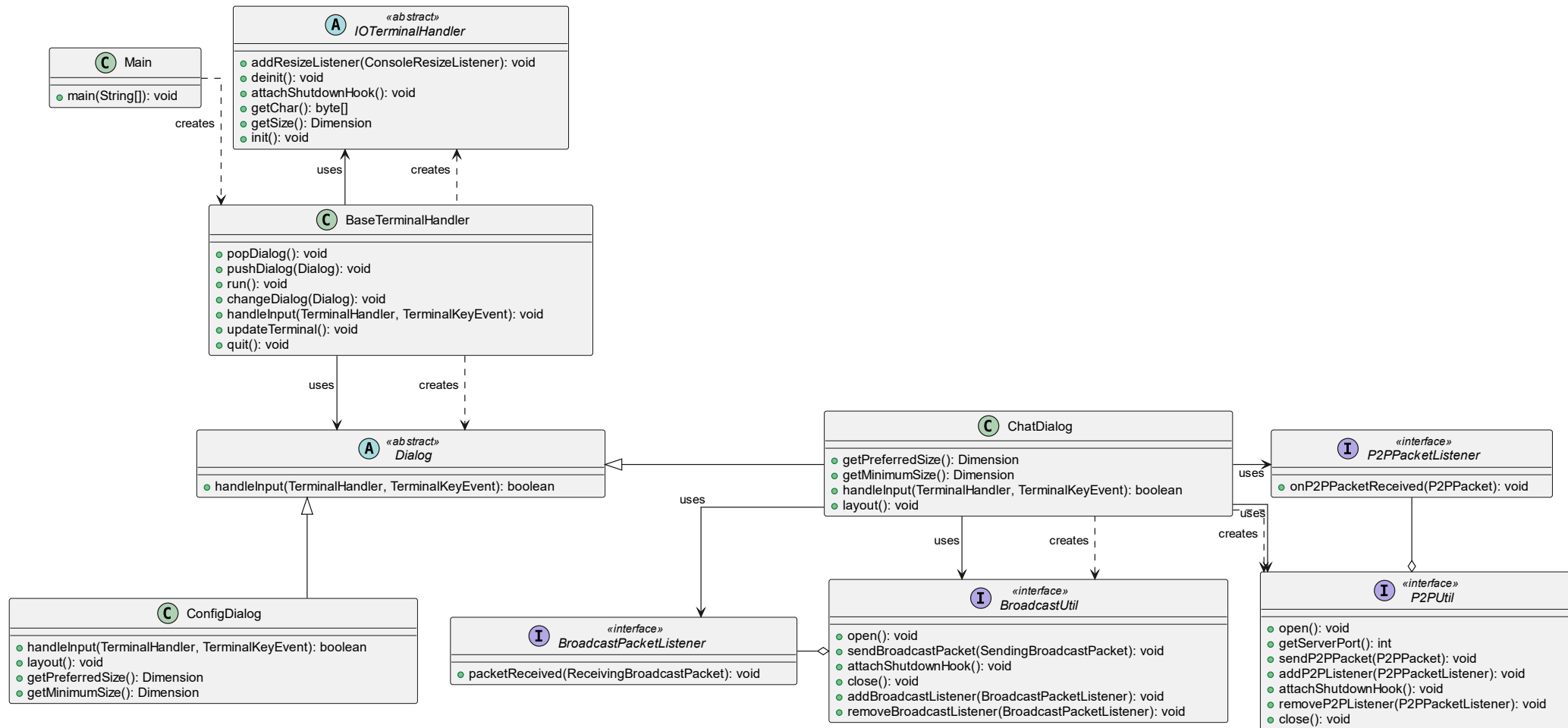
Einführung – Technischer Überblick

- Programmiersprache: Java
- Build Tool: Gradle

Softwarearchitektur

- Layered Architektur
 - Nicht Clean Architektur, da wir die Daten direkt in der UI liegen und nicht über die Domain-Logik abstrahiert sind
 - Mehrere Schichten, z.B. Rendering-Code, Networking, Config, ...
- Vorteile:
 - Schnell zu entwickeln
 - Intuitiv
- Nachteile:
 - Daten sind direkt in der UI --> nicht abstrahiert
 - Mehrere God-Classes, die alle Referenzen halten, somit sind Änderungen schwerer

Softwarearchitektur - UML



Softwarearchitektur – Domain Code

Domain Code :

- Enthält Grundlogik der Domäne
- Modelliert die Wirklichkeit der Domäne

```
tcp2PUtil.addP2PListener(packet -> {  
    if(packet instanceof MessageSendP2PPacket messagePacket) {  
        Message message = messagePacket.getJsonData();  
  
        Optional<Chat> chat = chats.stream().filter(it -> it.getSender()  
            .getName().equals(message.getSender().getName())).findFirst();  
  
        chat.ifPresent(value -> value.addMessage(message));  
  
        updateTerminal();  
    }  
}
```

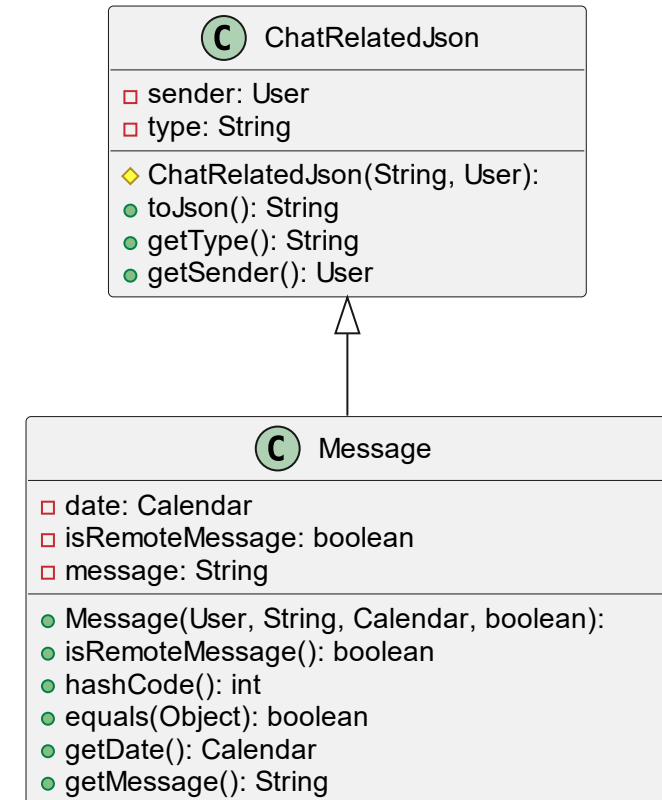
Dependency Rule - Positiv

- UI hält Referenzen auf die Entitäten wie z.B. User, Chats etc.
- Somit ist die UI von den Entitäten abhängig
 - Andersrum gilt das jedoch nicht



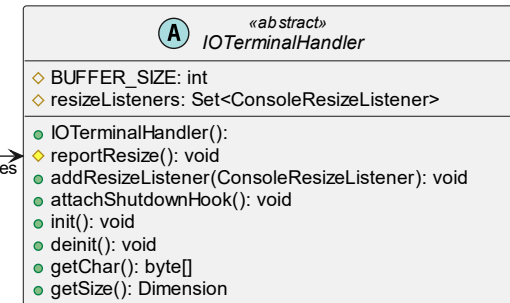
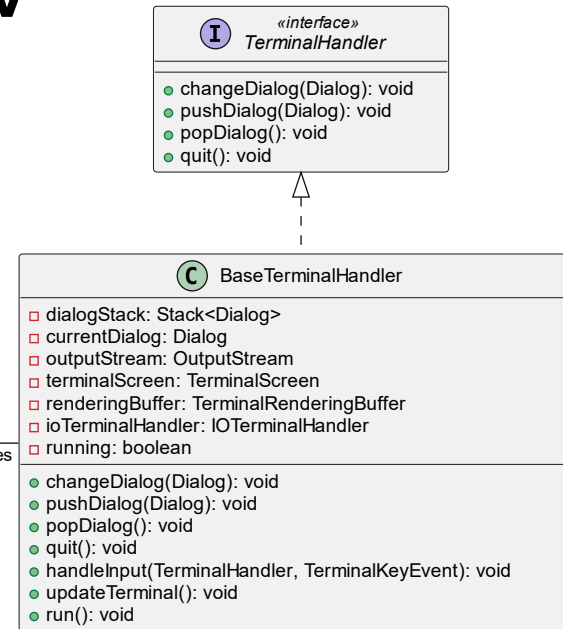
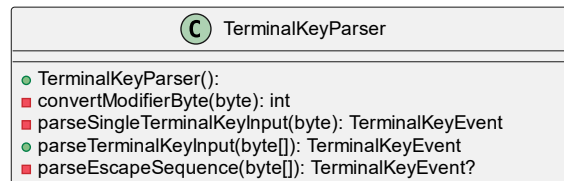
Dependency Rule - Negativ

- Entities wie z.B. Message oder Chat erweitern Code aus Networking
 - ChatRelatedJson gehört zur Networking-Schicht, da es für das Senden von Json-Packets verwendet wird
- Damit gehen die Abhängigkeiten aus einer niedrigen Schicht in einer höheren Schicht

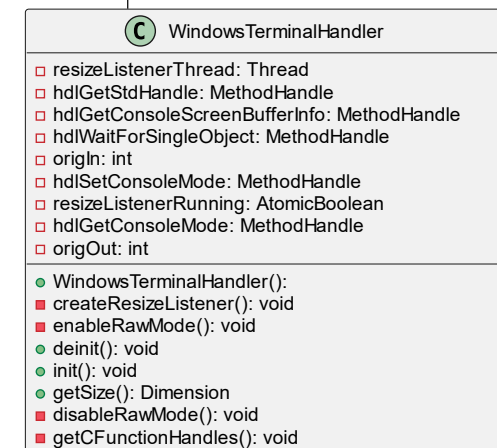
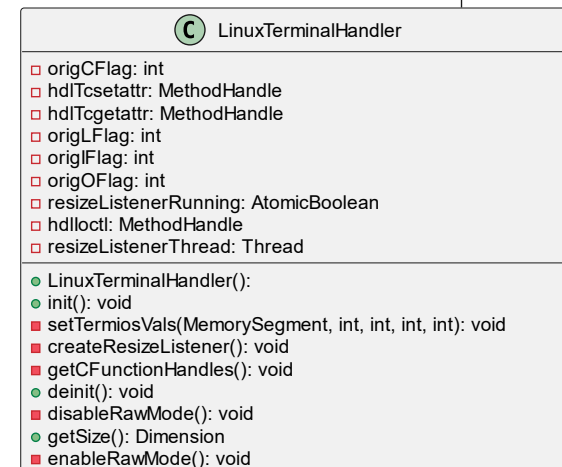


SOLID – SRP: Positiv

- Parsing und Input-Reading sind getrennt



- Ermöglicht einzelnes Testen

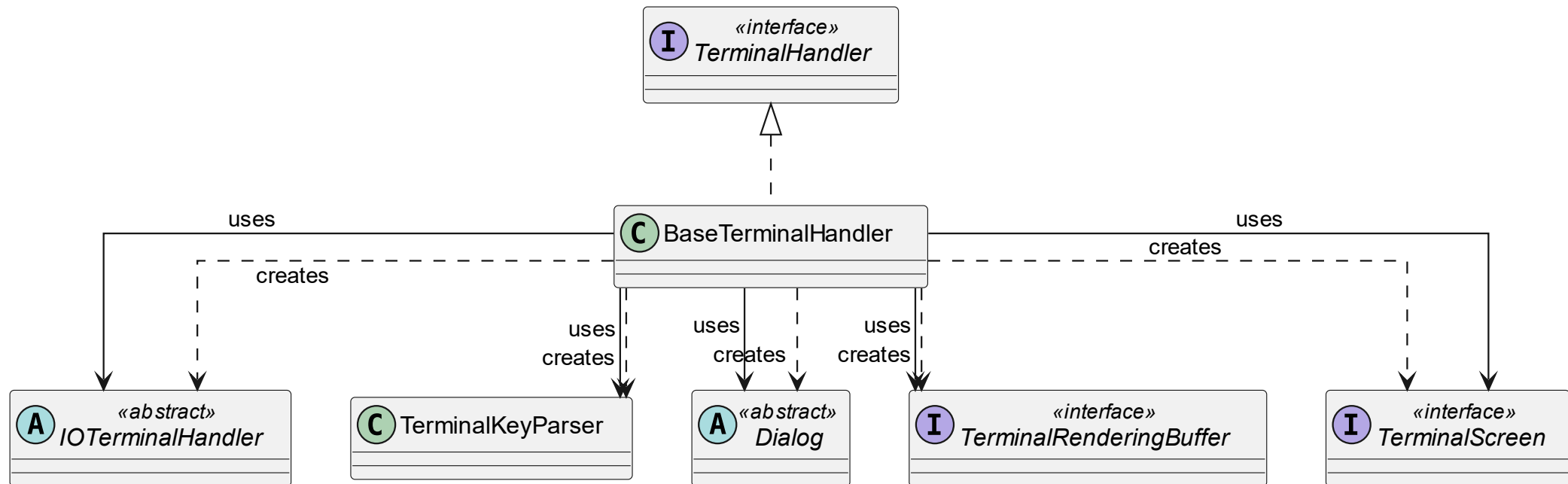


SOLID – SRP: Negativ

BaseTerminalHandler:

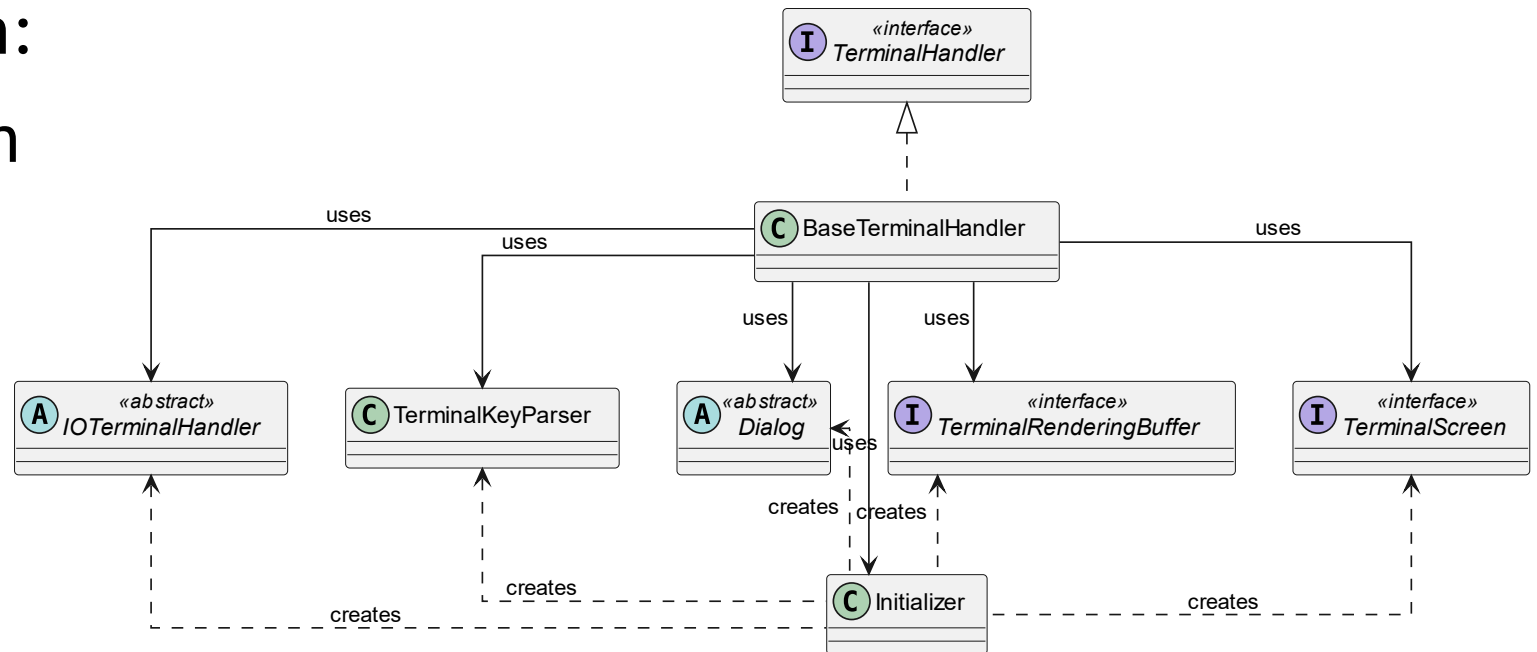
- Initialisierung aller Komponenten
- Layouten + Rendern
- Inputhandling

SOLID – SRP: Negative



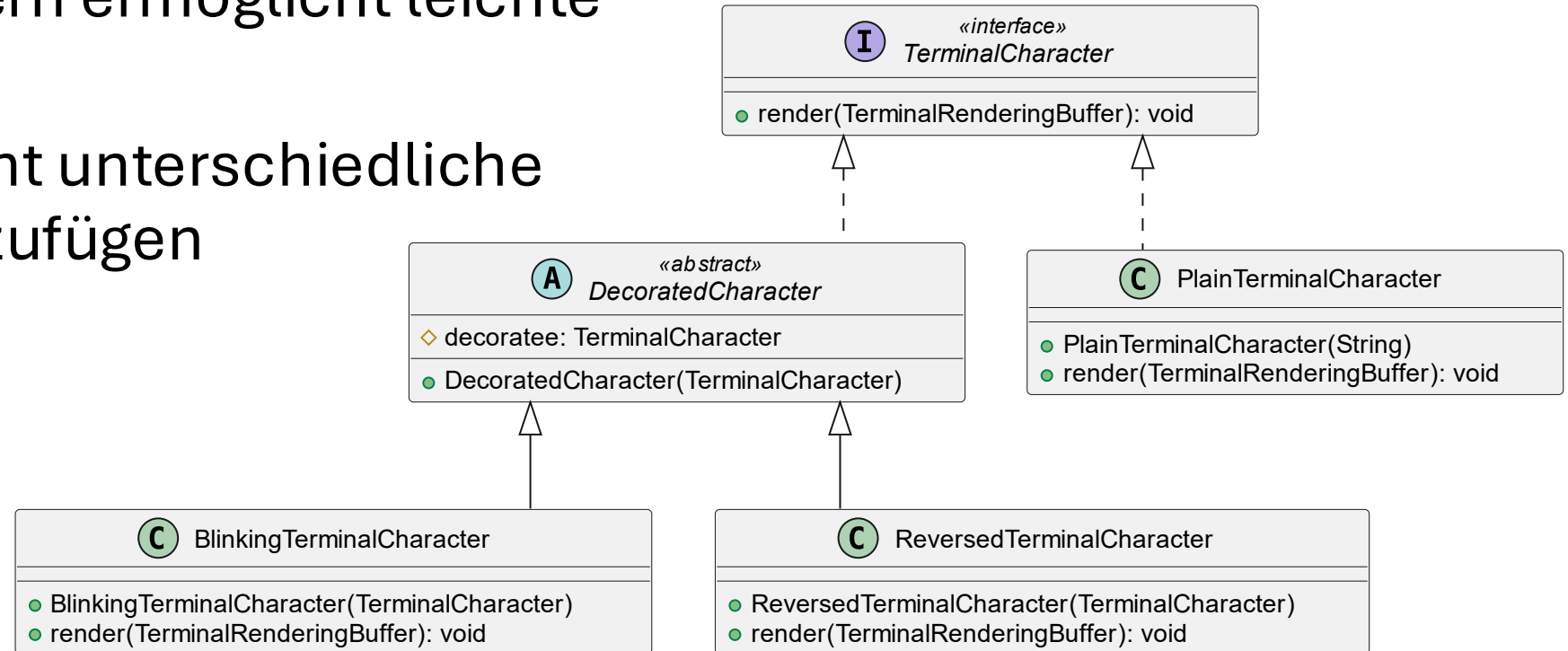
SOLID – SRP: Negativ

- Einführen eines Initializers
- Aufgaben nur noch:
- Layouten + Rendern
 - Inputhandling



SOLID – OCP: Positiv

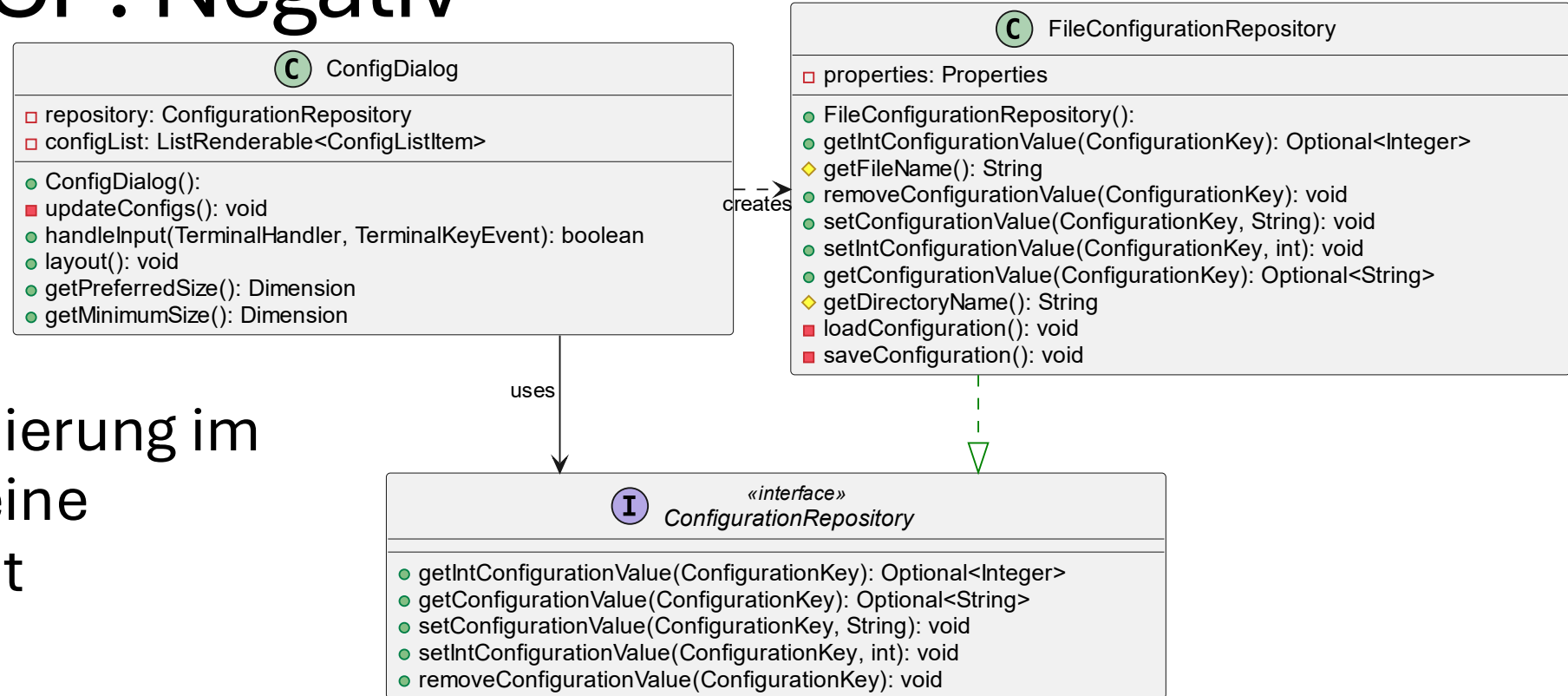
- Decorator Pattern ermöglicht leichte Erweiterung
- Ermöglicht leicht unterschiedliche Attribute hinzuzufügen



SOLID – OCP: Negativ

ConfigDialog:

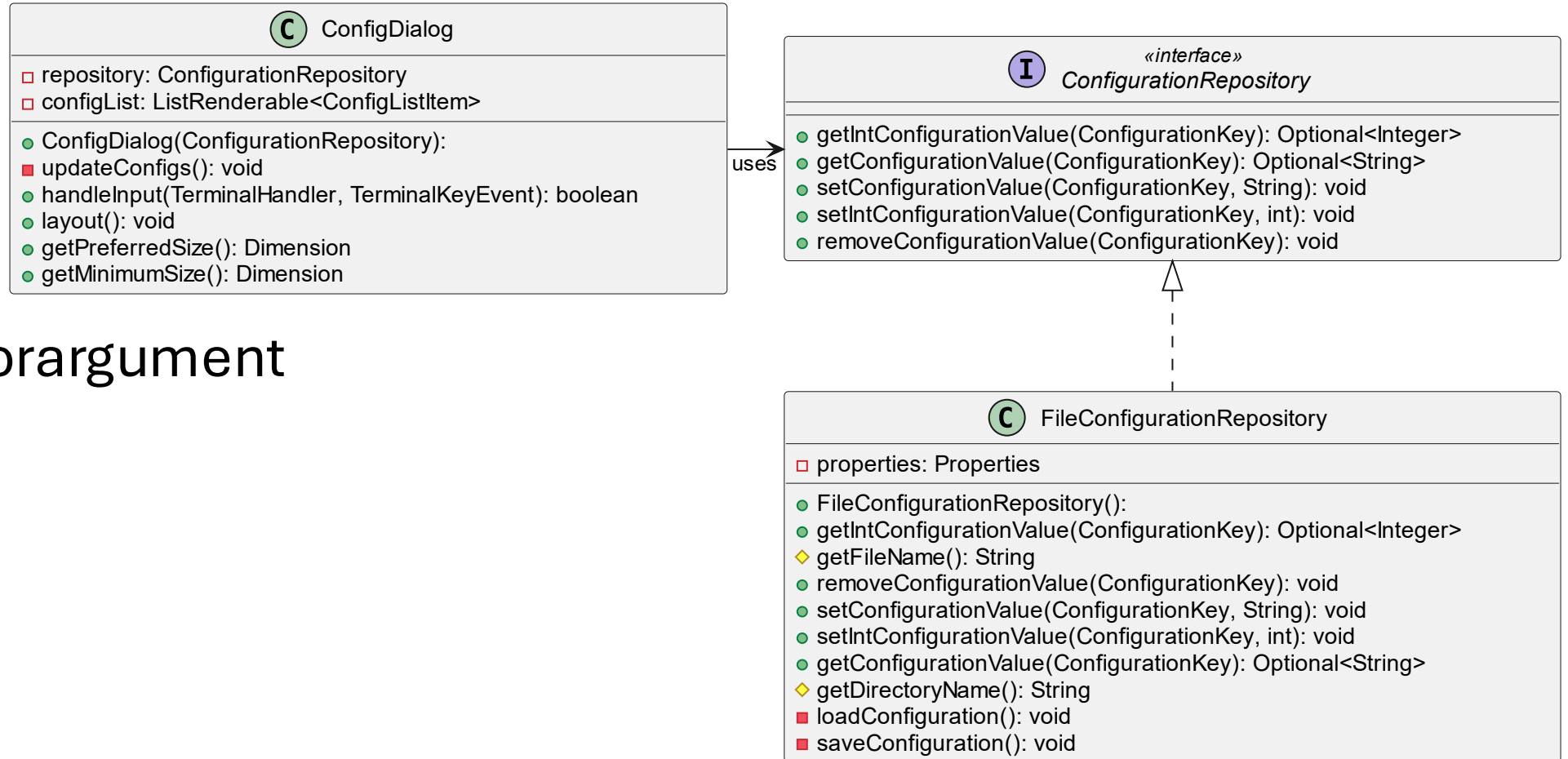
- Durch Instanziierung im Konstruktor keine Erweiterbarkeit



SOLID – OCP: Negativ

Lösung:

- Konstruktorargument

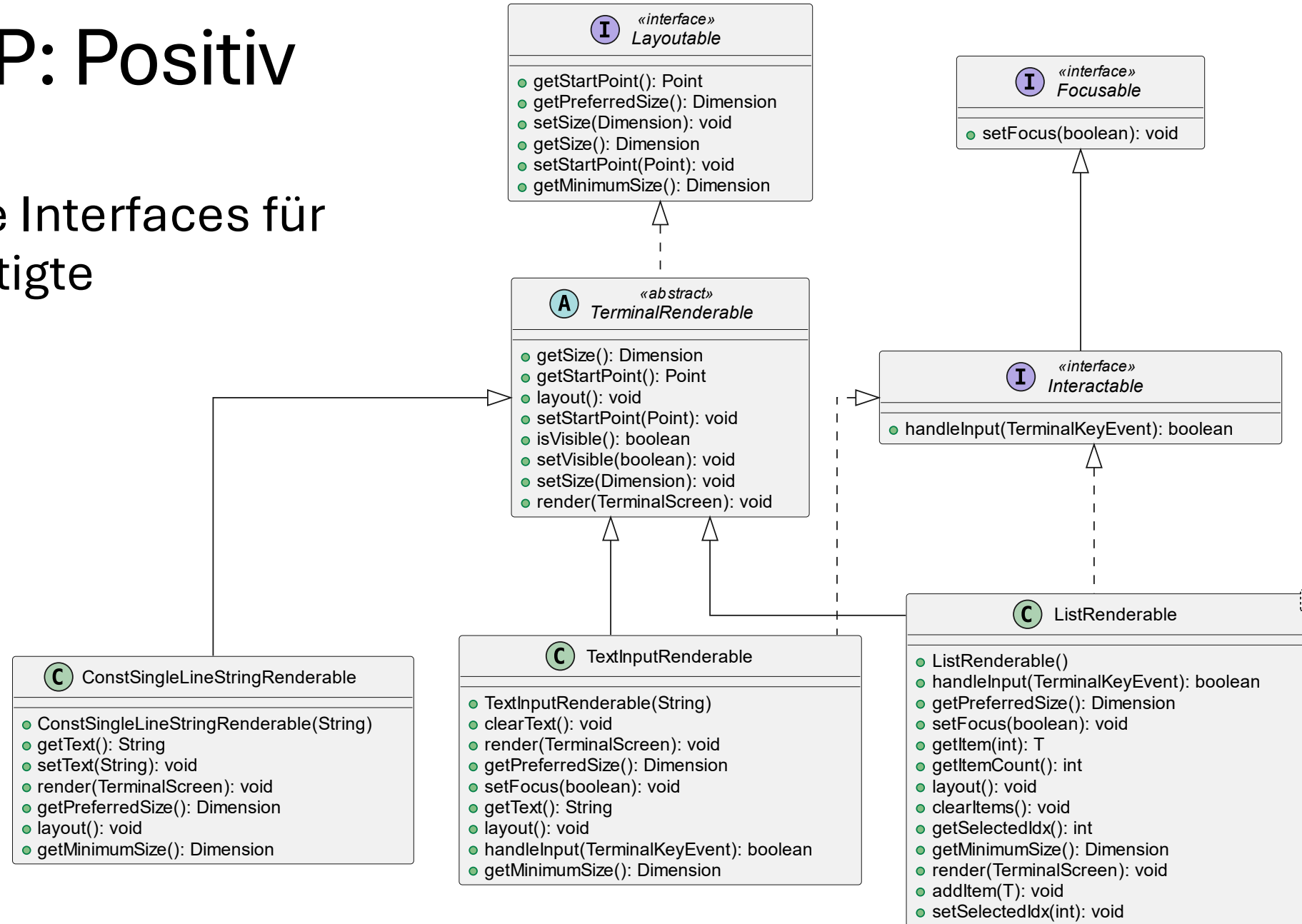


SOLID – ISP: Positiv

- Einzelne kleine Interfaces für jeweilige benötigte Funktionalität

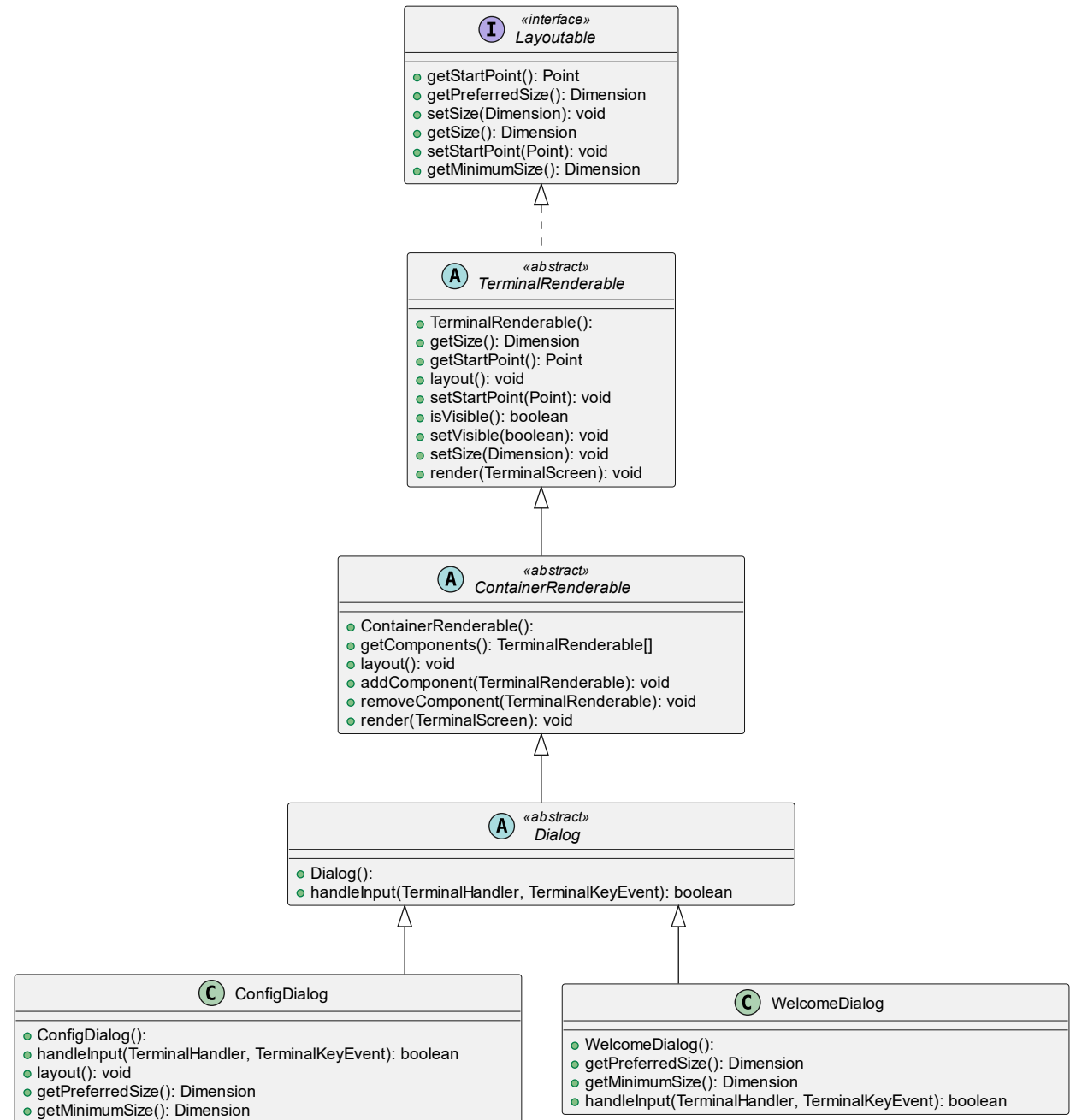
→ Modularer

→ Wartbarer



SOLID – ISP: Negativ

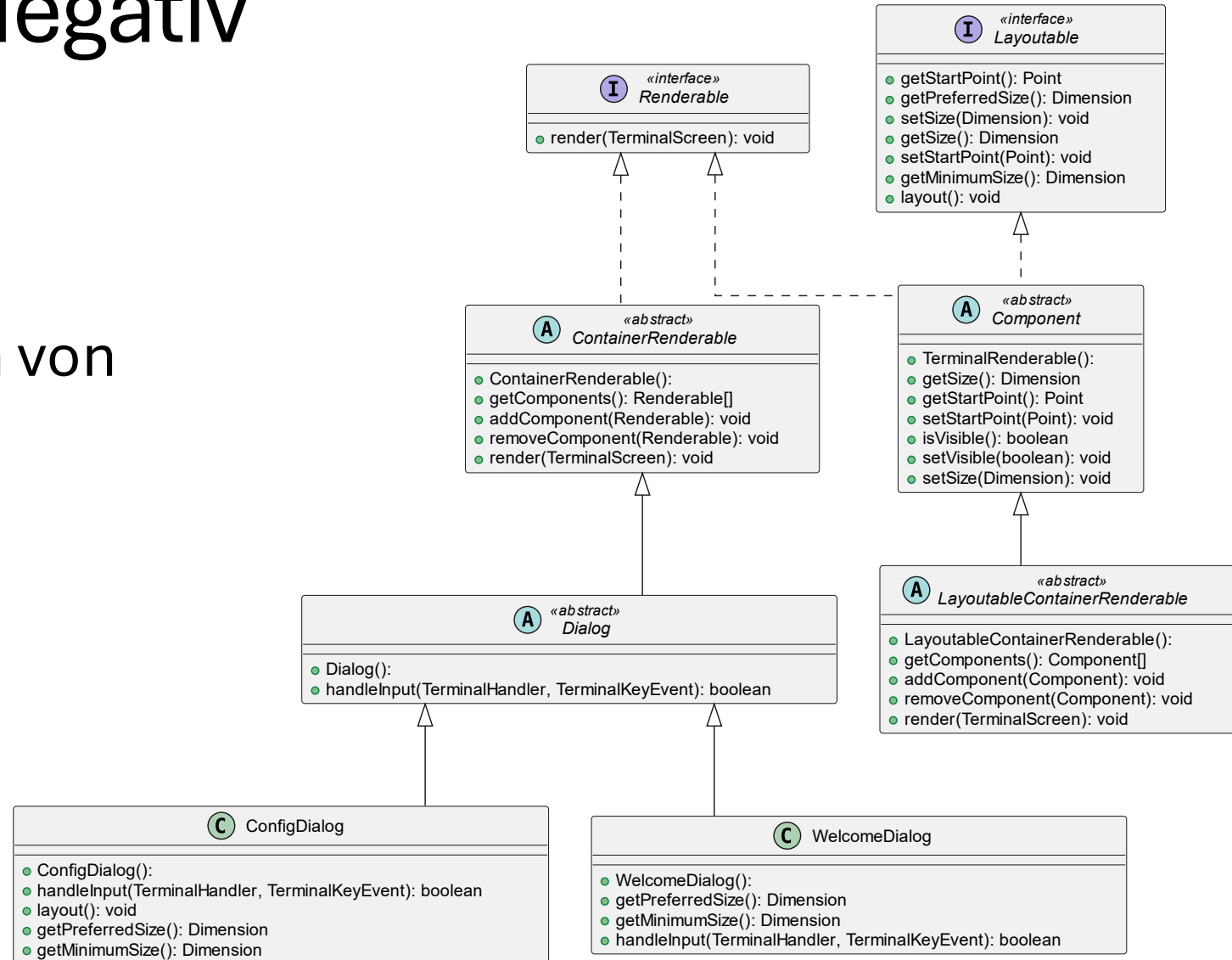
- Problem:
- Dialoge haben keine Size
- Dialoge sind Fullscreen



SOLID – ISP: Negativ

Lösung:

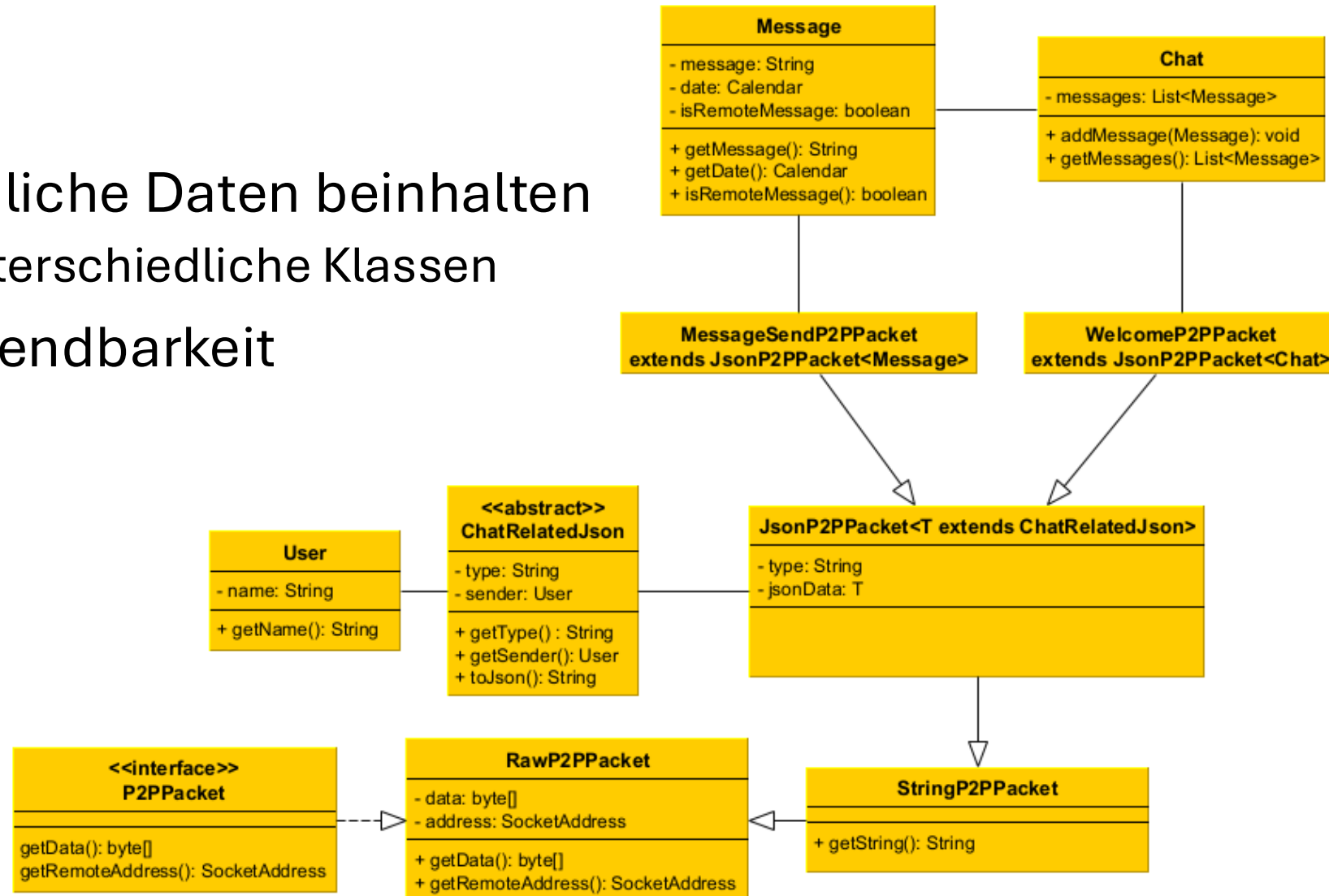
Einführung weiterer
Interfaces/ Extraktion von
Renderable



GRASP – Geringe Kopplung

Peer-To-Peer Packets:

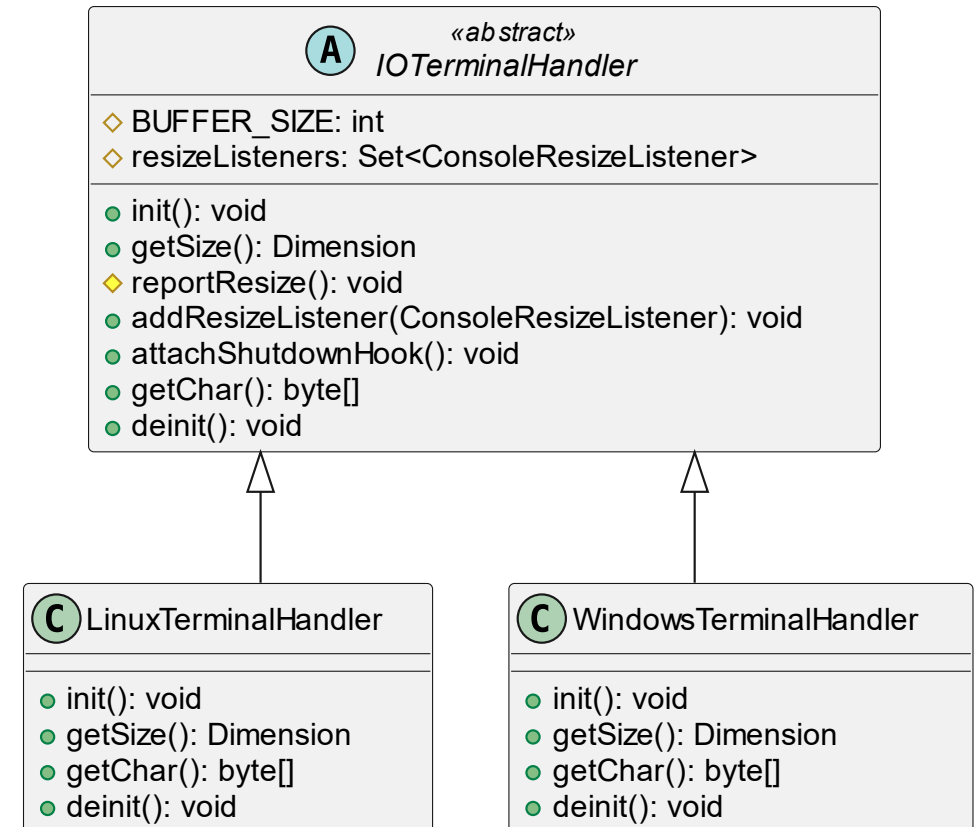
- Können unterschiedliche Daten beinhalten
 - z.B. String, Json, unterschiedliche Klassen
- Höhere Wiederverwendbarkeit
- Leichter anpassbar
- Bessere Lesbarkeit



GRASP: Polymorphismus

IOTerminalHandler:

- OS-spezifisch
- Besser Erweiterbar



DRY 1 - Commit:

a6095223b75966203ecc195f0deeee6a8f4ca96ae

- Problem/Doppelter Code:
 - In den IOTerminalHandlern gleiche Methode mit gleichem Code
- Extraktion der Methode in Parentclass

```
@Override
public byte[] getChar() {
    byte[] buffer = new byte[BUFFER_SIZE];

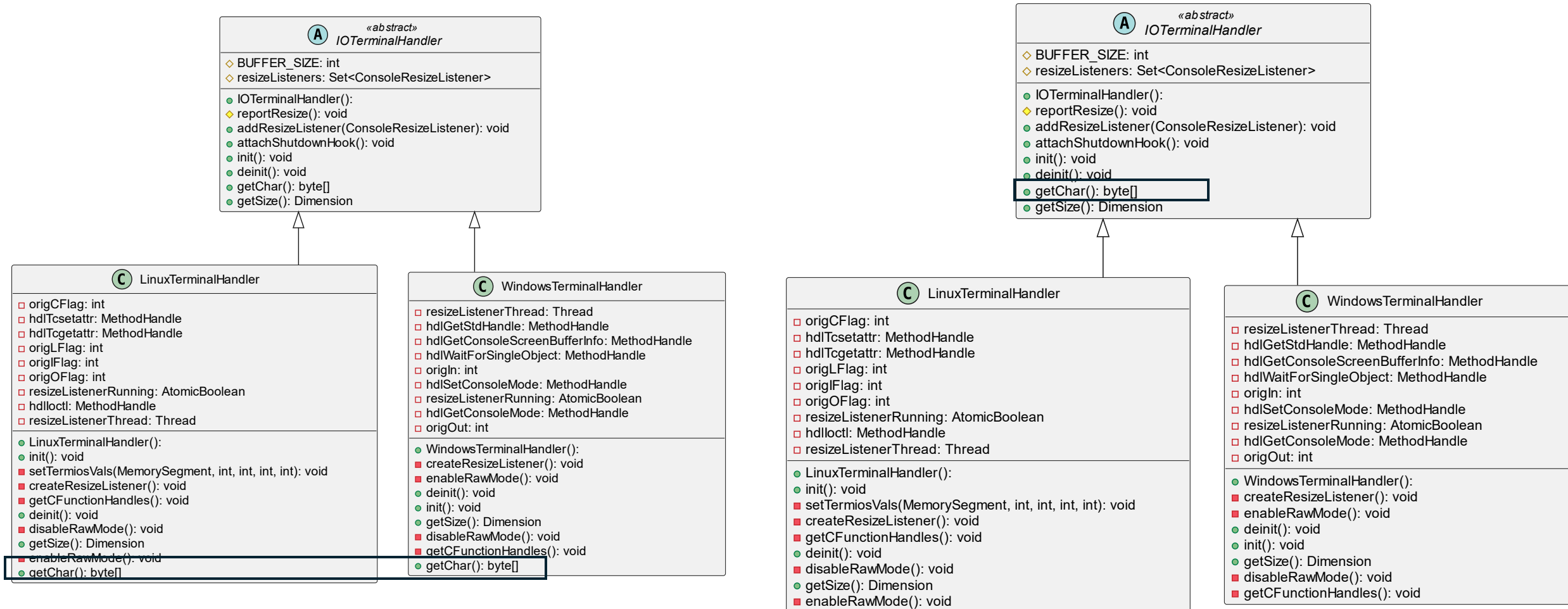
    int numRead = 0;
    try {
        numRead = System.in.read(buffer);
    } catch (IOException e) {
        return new byte[0];
    }

    // probably eof signal or kill
    if(numRead <= -1)
        return new byte[0];

    byte[] ret = new byte[numRead];
    System.arraycopy(buffer, 0, ret, 0,
numRead);
    return ret;
}
```

DRY 2 - Commit:

a6095223b75966203ecc195f0deee6a8f4ca96ae



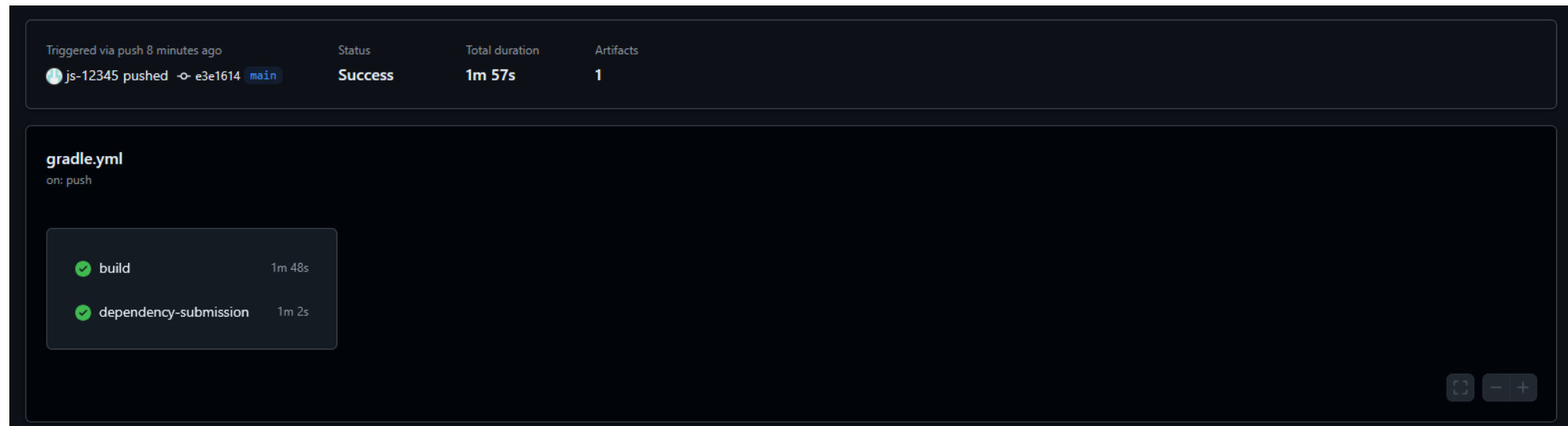
Unit Tests - Übersicht

Name	Beschreibung
testSingleAsciiSuccess	Testet einfaches ASCII-Parsing
testSingleAsciiAltModifierSuccess	Testet ASCII-Parsing mit Alt Modifier
testSingleAsciiError	Testet ASCII-Parsing mit Fehlerhafter Eingabe
testMultiAsciiSpecialSuccess	Testet ANSI-Special-Key-Parsing
testMultiAsciiAllModifierSuccess	Testet ANSI-Special-Key-Parsing mit Modifiern
testSingleUnicodeSuccess	Testet Unicode-Parsing
testSingleUnicodeAltModifierSuccess	Testet Unicode-Parsing mit Modifiern
testUnicodeStringSuccess	Testet Unicode-String-Parsing
testInitWithQExit	Testet einfacher Start und Ende durch IOTermianlHandlerMock
testBasicDialogRendering	Testet zum Start, ob Dialoge richtig gerendert werden durch EmptyDialogMock

Unit Tests - ATRIP

Automatic durch:

- GitHub Pipeline
- Lokaler Start der Tests durch IntelliJ



Unit Tests - ATRIP

Professional:

- Eindeutige Namensgebung
- Einhaltung von Code-Qualitätsstandards
- Vermeidung von Duplikaten
- Tests sind kurz und eindeutig



























Unit Tests - ATRIP

Thorough:

- Normale Eingaben werden getestet
- Grenzfälle werden getestet
- Fehler werden getestet

Unit Tests - ATRIP

aswe-chat-app

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
de.dhbw.ka.tinf22b5		0 %		n/a	2	2	4	4	2	2	1	1
de.dhbw.ka.tinf22b5.configuration		57 %		n/a	12	25	33	64	12	25	1	4
de.dhbw.ka.tinf22b5.net.broadcast		0 %		0 %	29	29	131	131	20	20	2	2
de.dhbw.ka.tinf22b5.net.broadcast.packets		0 %		n/a	3	3	6	6	3	3	1	1
de.dhbw.ka.tinf22b5.terminal.exception		0 %		n/a	2	2	4	4	2	2	1	1
de.dhbw.ka.tinf22b5.terminal.handler		94 %		100 %	2	12	8	62	2	9	0	1
de.dhbw.ka.tinf22b5.terminal.iohandler		76 %		33 %	4	9	7	24	2	7	0	2
de.dhbw.ka.tinf22b5.terminal.iohandler.lin		0 %		0 %	20	20	108	108	11	11	1	1
de.dhbw.ka.tinf22b5.terminal.iohandler.win		2 %		0 %	27	29	129	134	8	10	0	1
de.dhbw.ka.tinf22b5.terminal.key		26 %		13 %	206	237	203	265	6	23	1	4
de.dhbw.ka.tinf22b5.terminal.render		95 %		81 %	7	41	4	81	1	25	0	2
de.dhbw.ka.tinf22b5.terminal.render.characters		93 %		50 %	4	17	7	42	1	14	0	6
de.dhbw.ka.tinf22b5.terminal.render.components		58 %		35 %	75	127	96	261	11	55	1	8
de.dhbw.ka.tinf22b5.terminal.render.dialog		55 %		33 %	26	41	50	112	11	23	1	6
de.dhbw.ka.tinf22b5.terminal.render.layout		47 %		51 %	20	33	47	99	2	4	1	2
de.dhbw.ka.tinf22b5.util		76 %		30 %	6	9	7	20	1	4	0	3
Total	3.516 of 6.323	44 %	472 of 639	26 %	445	636	844	1.417	95	237	11	45

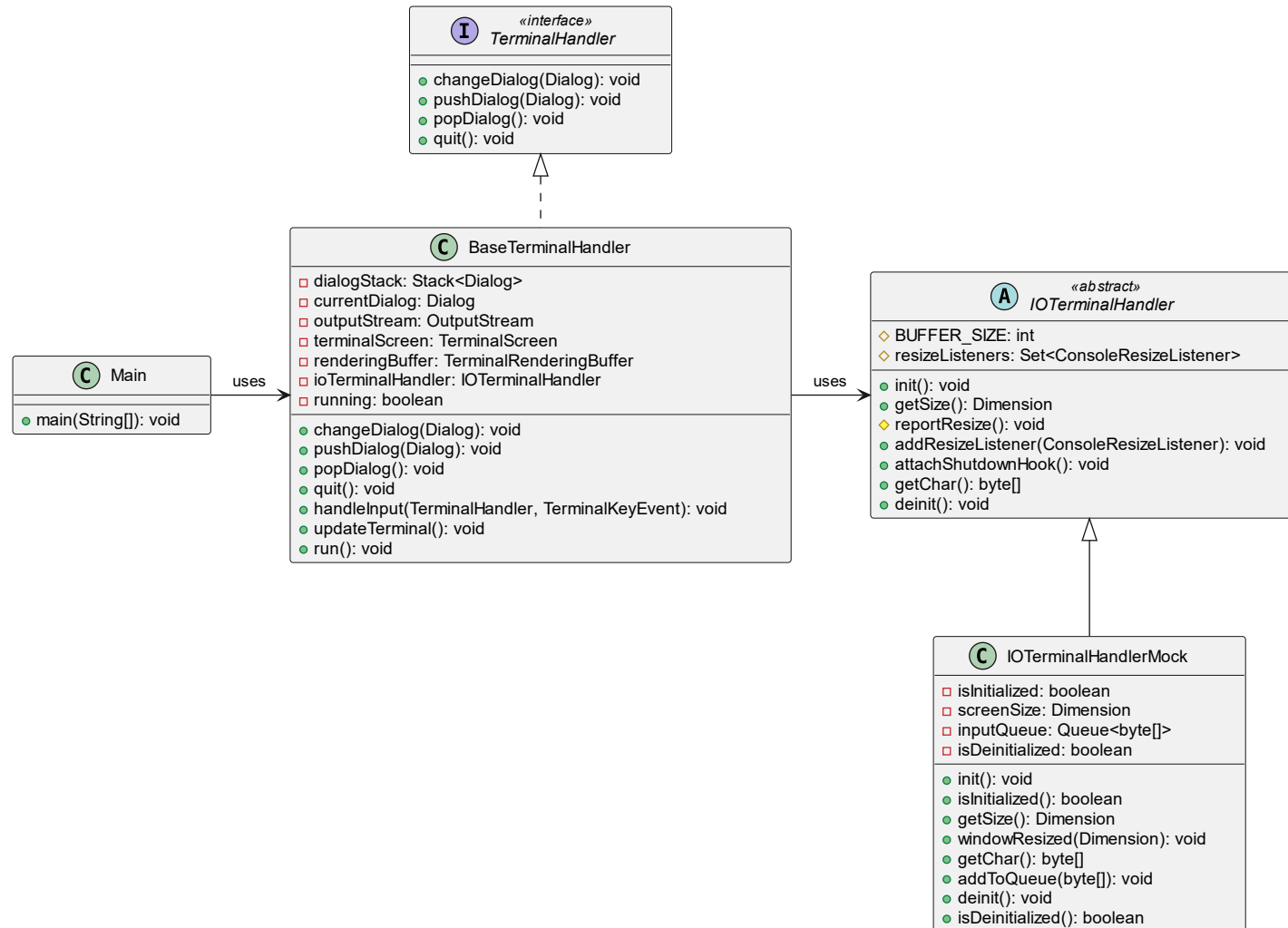
Unit Tests – IOTerminalHandlerMock 1

IOTerminalHandler zuständig für:

- Terminalinput
- Terminalgröße

→ Um Komponenten zu Testen, die auf Benutzereingabe reagieren benötigt es Mock

Unit Tests – IOTerminalHandlerMock 2



Unit Tests – EmptyDialogMock 1

Dialog zuständig für:

- Rendering des aktuellen Dialogs
- Layouting der Komponenten
- Input Handling

→ Um Komponenten zu Testen, die gerendert werden und die, die Dialoge rendern, benötigt es ein Mock

Unit Tests – EmptyDialogMock 2



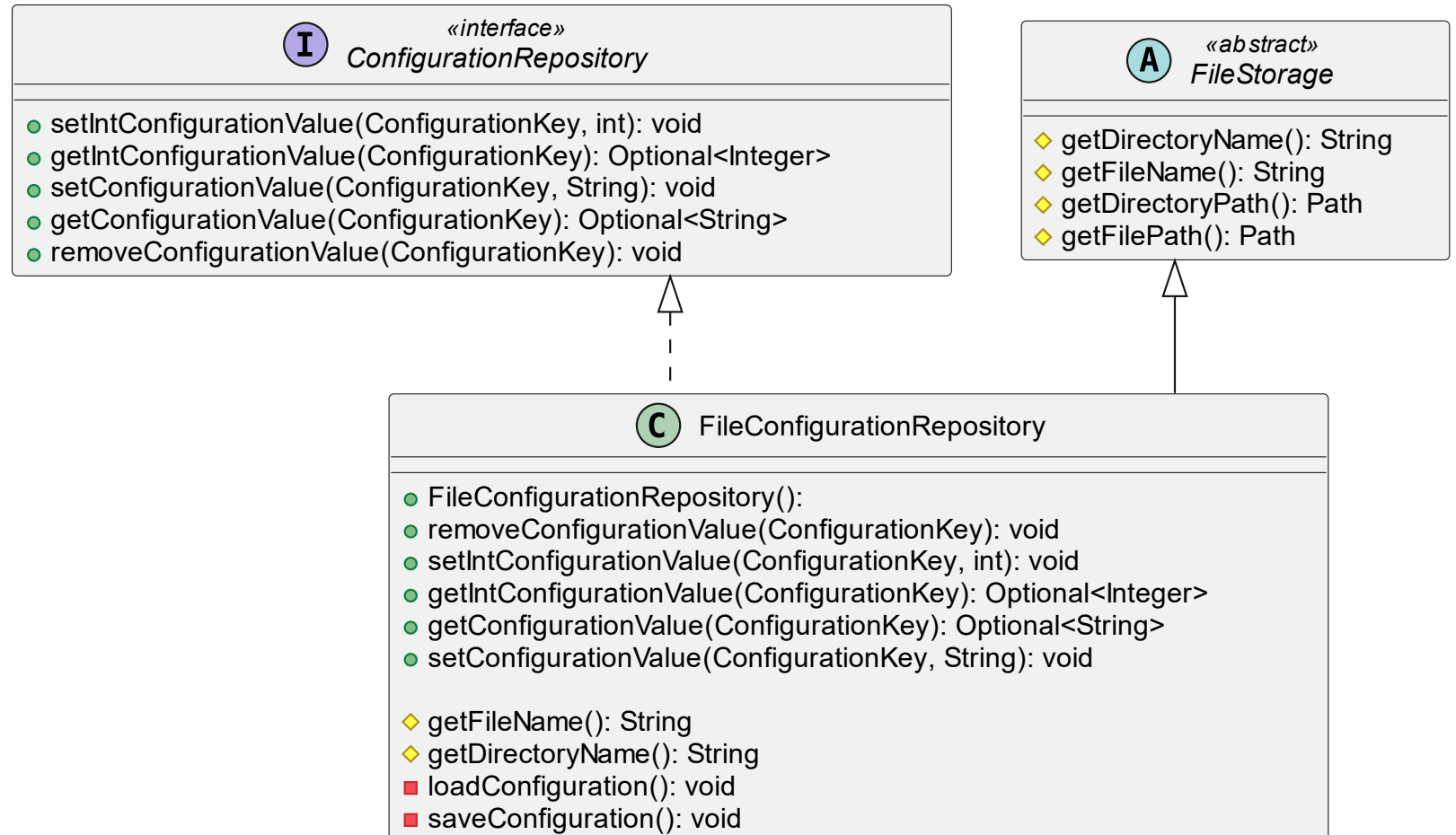
Domain Driven Design 1

Ubiquitous Language

Bezeichnung	Bedeutung	Begründung
User	Benutzer der App	Grundlegender Akteur
Message	Nachricht eines Benutzers zu einem anderen	Grundlegendes Objekt
Chat	Nachrichtenverlauf zwischen zwei Benutzern	Grundlegende Ansammlung der Messages und Usern
Connection	Verbindung zwischen zwei Usern, um Nachrichten zu verschicken	Grundlegende Funktionalität einer Chat App

Domain Driven Design – Repository

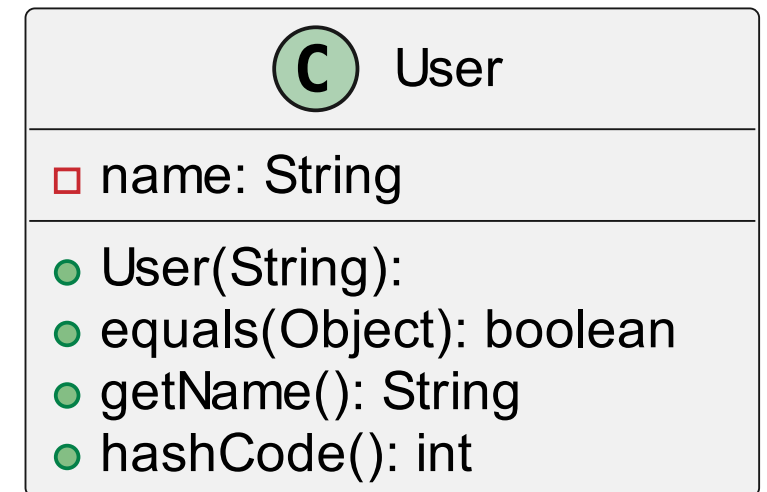
- Lädt die Konfiguration der App
- Abstrahiert die Speicherung der Konfiguration und ermöglicht einfache Erweiterungen



Domain Driven Design – Entity

User:


- Beschreibt einen eindeutigen Benutzer
- Persistente Identität
- Wird benötigt für Erweiterungen



Domain Driven Design – Value Object

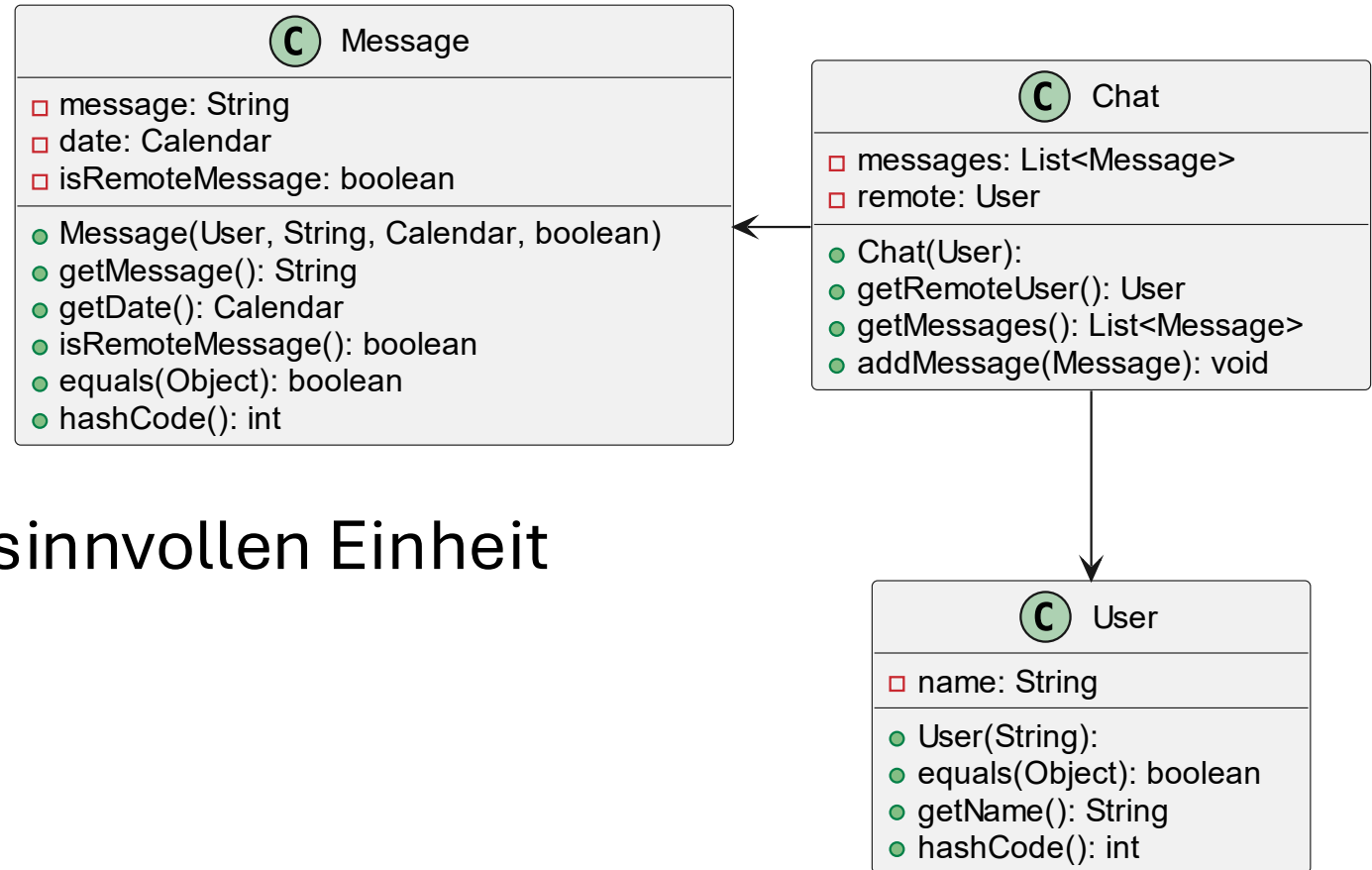
Message:

- abstrahiert Details
- klareres, verständlicheres Modell
- Sicherheit durch Unveränderlichkeit

 Message
<ul style="list-style-type: none">❑ message: String❑ date: Calendar❑ isRemoteMessage: boolean
<ul style="list-style-type: none">● Message(User, String, Calendar, boolean)● getMessage(): String● getDate(): Calendar● isRemoteMessage(): boolean● equals(Object): boolean● hashCode(): int

Domain Driven Design – Aggregate

Chat:



- Kapselt Messages zu einer sinnvollen Einheit
- Regelt Zugriff

Code Smell 1 – Method Chains

- Lange Aufrufkette mehrfach wiederholt

Lösung: Auslagerung in Chat-Klasse

Vorher	Nachher
<pre>Chat.getMessage().addFirst(message);</pre>	<pre>Chat.addMessage(message); public void addMessage(Message message) { messages.addFirst(message); }</pre>

Code Smell 2 – Duplicated Code

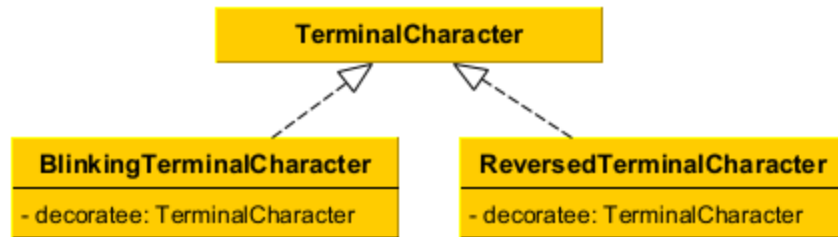
- Langer Code zur Erzeugung von Gson mehrfach wiederholt
- Lösung: Auslagerung in Util-Klasse, welche Gson-Erzeugung übernimmt

Vorher	Nachher
<pre>new GsonBuilder() .registerTypeAdapter(ChatRelatedJson.class, new PolymorphDeserializer<ChatRelatedJson>()).create().toJson(this);</pre>	<pre>public static Gson getGson() { return new GsonBuilder() .registerTypeAdapter(ChatRelatedJson.class, new PolymorphDeserializer<ChatRelatedJson>()).create(); } GsonUtil.getGson().toJson(this);</pre>

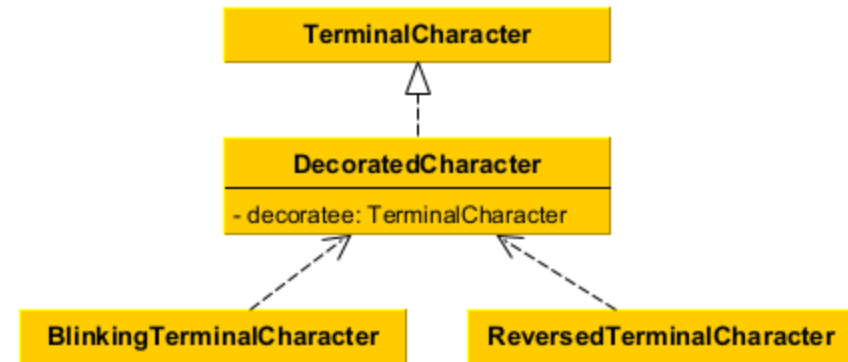
Refactoring 1 – Extract Class

- Commit ID: f4f51fe91f894d5c71d8bf5f7c6396b52f522aca
- Problem: Nicht alle TerminalCharacter benötigen einen Decorator
- Lösung: Extrahieren der Funktion in DecoratedCharacter

Vorher



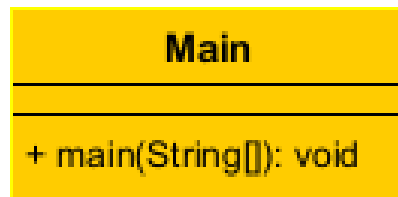
Nachher



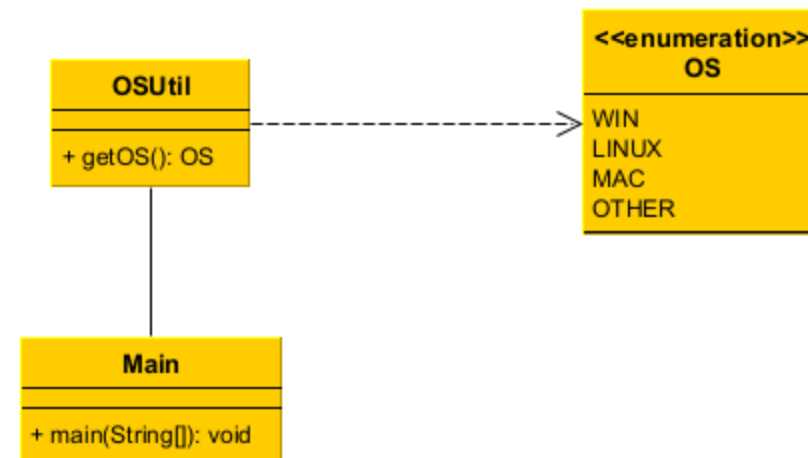
Refactoring 2 – Extract Method

- Commit ID: 08b1340371216d3d0aad8ab64e31bceda13713c9
- Problem: Herausfinden des Betriebssystems bei Start ist sehr lang
- Lösung: Extrahieren der Methode in eine eigene Util-Klasse

Vorher



Nachher



Refactoring 2 – Extract Method – Vorher

```
public static void main(String[] args) throws IOException, TerminalHandlerException {  
    BaseTerminalHandler terminal;  
    if (System.getProperty("os.name").toLowerCase().contains("windows")) {  
        terminal = new WindowsTerminalHandler(new WelcomeDialog());  
    } else {  
        terminal = new LinuxTerminalHandler(new WelcomeDialog());  
    }  
}
```

Refactoring 2 – Extract Method – Nachher

```
public class OSUtil {  
    public static OS getOS() {  
        String osName = System.getProperty("os.name", "other").toLowerCase();  
        if (osName.contains("mac") || osName.contains("darwin")) {  
            return OS.MAC;  
        } else if (osName.contains("windows")) {  
            return OS.WIN;  
        } else if (osName.contains("nux") || osName.contains("nix")) {  
            return OS.LINUX;  
        } else {  
            return OS.OTHER;  
        }  
    }  
}
```

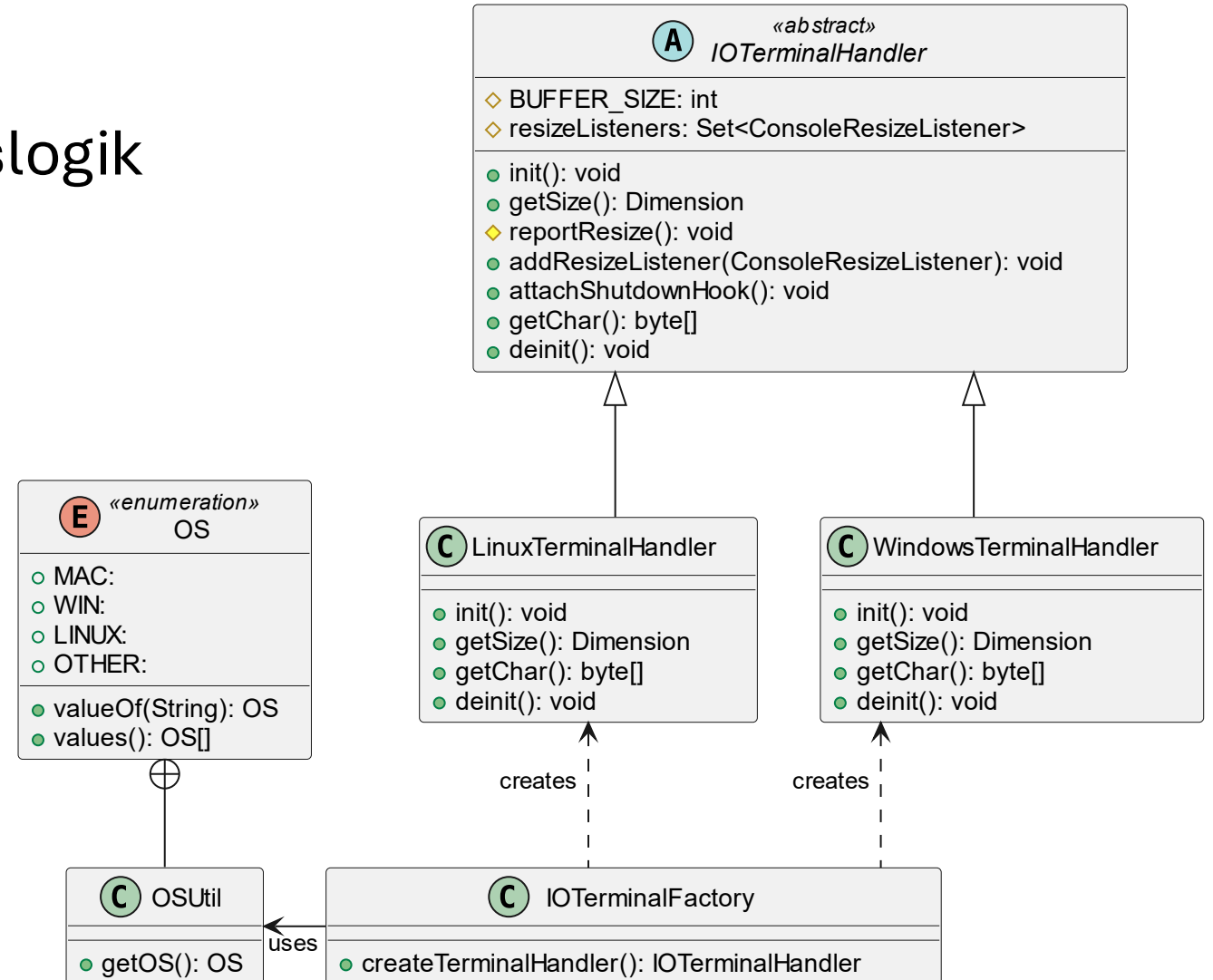
Refactoring 2 – Extract Method – Nachher

```
public enum OS {  
    WIN,  
    LINUX,  
    MAC,  
    OTHER;  
}
```

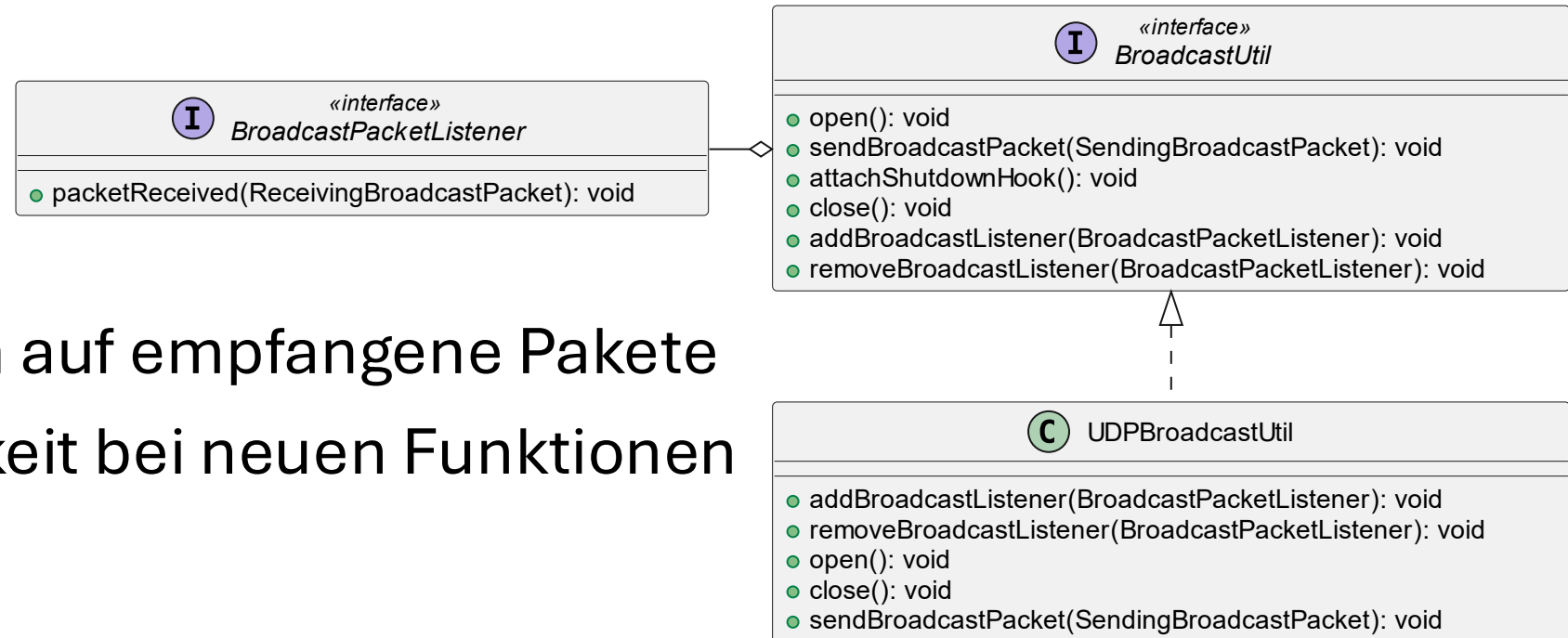
```
public static void main(String[] args) throws IOException, TerminalHandlerException {  
    BaseTerminalHandler terminal = switch (OSUtil.getOS()) {  
        case WIN -> new WindowsTerminalHandler(new WelcomeDialog());  
        case LINUX -> new LinuxTerminalHandler(new WelcomeDialog());  
        default -> throw new RuntimeException("Unsupported OS: " + OSUtil.getOS());  
    };  
}
```

Entwurfsmuster - IOTerminalFactory

- Komplizierte Instanziierungslogik
- Bessere Erweiterbarkeit



Entwurfsmuster – Observer/Listener



- Einfache Reaktion auf empfangene Pakete
- Gute Erweiterbarkeit bei neuen Funktionen