

# Programmentwurf

## Fun & Learn

Name: Rouven Gremsperger und Jennifer Vanessa Zaminer  
Matrikelnummer: 5146804 und 4191630

Abgabedatum: [19.04.2023]

## Allgemeine Anmerkungen:

- *es darf nicht auf andere Kapitel als Leistungsnachweis verwiesen werden (z.B. in der Form “XY wurde schon in Kapitel 2 behandelt, daher hier keine Ausführung”)*
- *alles muss in UTF-8 codiert sein (Text und Code)*
- *sollten mündliche Aussagen den schriftlichen Aufgaben widersprechen, gelten die schriftlichen Aufgaben (ggf. an Anpassung der schriftlichen Aufgaben erinnern!)*
- *alles muss ins Repository (Code, Ausarbeitung und alles was damit zusammenhängt)*
- *die Beispiele sollten wenn möglich vom aktuellen Stand genommen werden*
  - *finden sich dort keine entsprechenden Beispiele, dürfen auch ältere Commits unter Verweis auf den Commit verwendet werden*
  - *Ausnahme: beim Kapitel “Refactoring” darf von vorne herein aus allen Ständen frei gewählt werden (mit Verweis auf den entsprechenden Commit)*
- *falls verlangte Negativ-Beispiele nicht vorhanden sind, müssen entsprechend mehr Positiv-Beispiele gebracht werden*
  - *Achtung: werden im Code entsprechende Negativ-Beispiele gefunden, gibt es keine Punkte für die zusätzlichen Positiv-Beispiele*
  - *Beispiele*
    - *“Nennen Sie jeweils eine Klasse, die das SRP einhält bzw. verletzt.”*
      - *Antwort: Es gibt keine Klasse, die SRP verletzt, daher hier 2 Klassen, die SRP einhalten: [Klasse 1], [Klasse 2]*
      - *Bewertung: falls im Code tatsächlich keine Klasse das SRP verletzt: volle Punktzahl ODER falls im Code mind. eine Klasse SRP verletzt: halbe Punktzahl*
- *verlangte Positiv-Beispiele müssen gebracht werden*
- *Code-Beispiel = Code in das Dokument kopieren*

# 1: Einführung

## Übersicht über die Applikation

[Was macht die Applikation? Wie funktioniert sie? Welches Problem löst sie/welchen Zweck hat sie?]

Die Applikation Fun&Learn (<https://github.com/0Raspbinary1/Advanced-SE>) ist eine Anwendung, die es dem Benutzer ermöglicht zwischen verschiedenen Kategorien zu entscheiden wie „Spielen“, „Quiz“, „Umrechnen“ oder „Notizen“. In der Kategorie „Spielen“ kann der Benutzer zwischen zwei Spielen „Zahlen raten“ und „Schere Stein Papier“ auswählen. In der Kategorie „Quiz“ kann der Benutzer zwischen zwei Quiz-Kategorien auswählen (Deutschland-Quiz und Astronomie-Quiz). In der Kategorie „Umrechnen“ kann der Benutzer verschiedene Werte umrechnen wie beispielsweise Gewicht oder Temperatur. In der Kategorie „Notizen“ hat der Benutzer die Möglichkeit eine Notiz zu erstellen, zu löschen und anzusehen. Die Applikation Fun&Learn soll den Zweck erfüllen, Spaß zu haben mit den Spielen, jedoch auch die Möglichkeit haben, mit dem Quiz und den Notizen, etwas zu lernen und produktiv zu sein. Des Weiteren stellt sie mit dem Tool des Umrechnens eine Hilfestellung beziehungsweise ein Nachschlagewerk bereit. Da dies alles in einer Anwendung zur Verfügung steht kann ein Benutzer die verschiedenen Systeme in einem gebündelt und auf einer Konsole verwenden. Dafür werden über Konsoleneingaben die Menüpunkte ausgewählt und der Benutzer kann so durch das Programm navigieren und die verschiedenen Optionen verwenden.

## Wie startet man die Applikation?

[Wie startet man die Applikation? Welche Voraussetzungen werden benötigt? Schritt-für-Schritt-Anleitung]

Es sollte Java auf dem Rechner installiert sein im Moment verwendet das Projekt Java 11, das kann aber beliebig über den Buildpath oder die Pom.xml Datei angepasst werden. Zusätzlich wird falls die Jar-Datei des gesamten Projekts mit Maven erzeugt werden soll eine lokale Maven-Installation benötigt.

Für das Ausführen der Anwendung hier noch eine Schritt für Schritt Anleitung:

1. Repo von Github klonen oder den Code als Zip herunterladen
2. Projekt kompilieren indem eine der Möglichkeiten gewählt wird, sollte die Anwendung direkt über eine IDE gestartet werden fällt dieser Schritt weg
  - die Klasse Fun&Learn im Package „main.java.all“ mit der Konsole über javac <Dateiname> kompilieren
  - mit Maven eine jar Datei erzeugen indem der Befehl „mvn install“ ausgeführt wird
3. das eigentliche Starten der Anwendung unterscheidet sich je nachdem für welchen Weg sich unter 2. entschieden wurde

- existiert eine Jar-Datei kann diese nun ausgeführt werden und somit die Anwendung gestartet werden
  - wurde der unter 2. genannte Schritt übersprungen kann die Anwendung in einer IDE über das Ausführen der Klasse Fun&Learn im Package „main.java.all“ gestartet werden
4. Die App in der Konsole über die Menüs bedienen und mit den Spaß- und Lernfortschritten beginnen.

## Wie testet man die Applikation?

*[Wie testet man die Applikation? Welche Voraussetzungen werden benötigt? Schritt-für-Schritt-Anleitung]*

Die Anwendung enthält JUnit-Tests. Als Voraussetzungen für das Ausführen der Tests ist eine Installation von Java von Nöten, des Weiteren ist Maven oder eine lokal ins Projekt eingebundene Version von JUnit benötigt.

Für die Durchführung der Tests gibt es mehrere Möglichkeiten:

- mvn test bzw. bereits beim Erstellen einer Jar Datei mit mvn install
- manuelles Ausführen der Testklassen in den Verzeichnissen aus „src.test.java“
  - kompilieren der Klassen über javac und ausführen der Jar Datei
  - direktes Ausführen der Klassen in einer IDE

Bei Maven wird das Ergebnis der Analyse in einem Verzeichnis innerhalb eines Unterverzeichnisses des Target Verzeichnisses abgelegt.

## Kapitel 2: Clean Architecture

### Was ist Clean Architecture?

*[allgemeine Beschreibung der Clean Architecture in eigenen Worten]*

Bei dem Prinzip Clean Architecture handelt es sich um ein Architektur Prinzip für Software. Dessen primäres Ziel es ist, eine Kopplung der Komponenten so gering wie möglich aber so groß wie notwendig zu halten. Dabei existieren verschiedene Schichten diese untergliedern sich von außen nach innen in die Plugin-, Adapter-, Applikations-, Domainen Code- und Abstraktionsschicht. Dabei befinden sich die Codeaspekte, die eine konkrete und situationsabhängige Logik implementieren weiter außen. Ebenfalls zu diesem Prinzip gehört, dass der Anwendungscode und Domaincode nicht von anderem Code abhängt. Gekoppelt sind die äußeren Schichten an die inneren und implementieren dort definierte Interfaces. Durch das Auslagern der Abhängigkeiten ändert ein Wechsel der Technologien die Anwendung nicht, jedoch müssen die Adapter hierfür gegebenenfalls angepasst werden. In diesem Zuge findet auch eine Separation der Plugins statt.

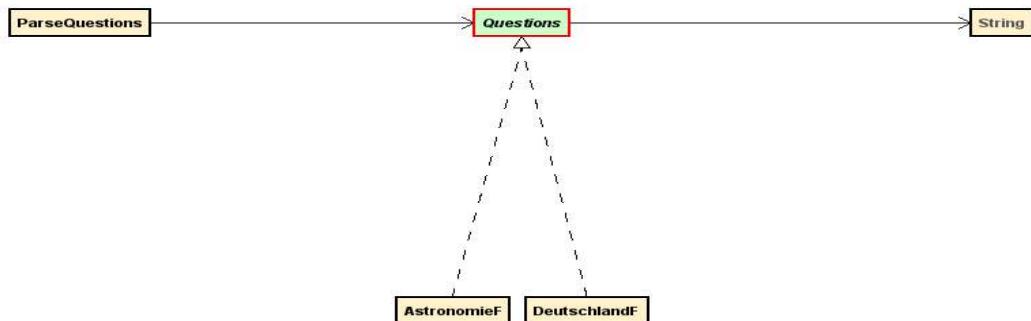
## Analyse der Dependency Rule

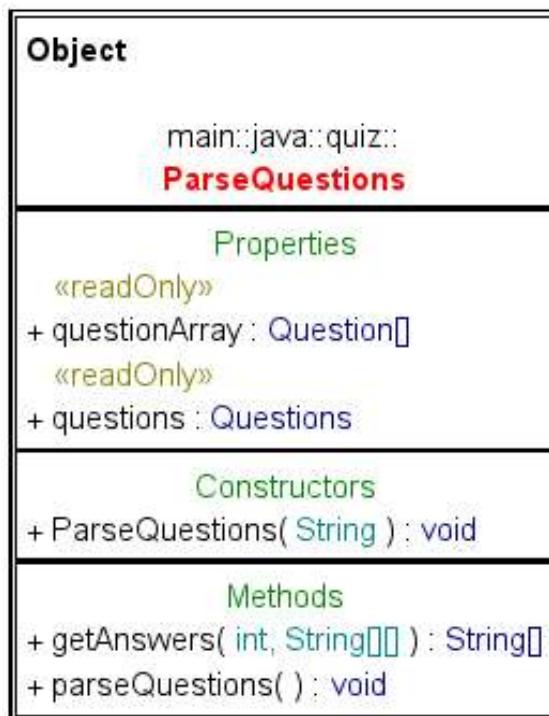
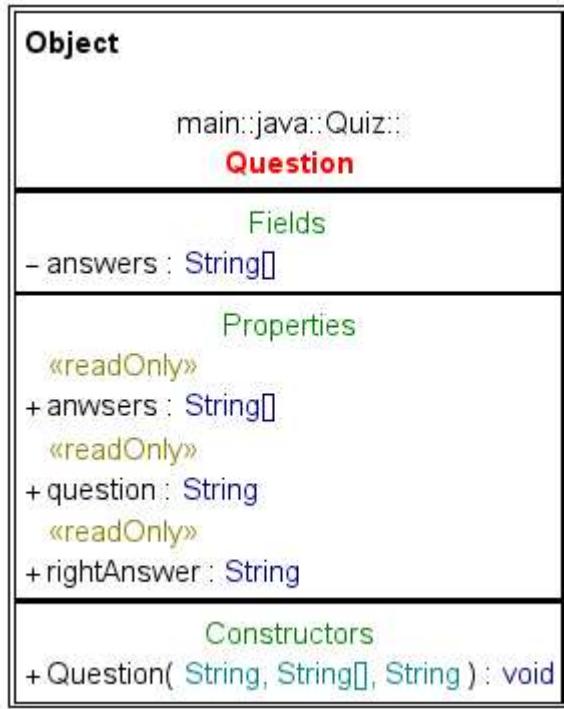
*[(1 Klasse, die die Dependency Rule einhält und eine Klasse, die die Dependency Rule verletzt); jeweils UML der Klasse und Analyse der Abhängigkeiten in beide Richtungen (d.h., von wem hängt die Klasse ab und wer hängt von der Klasse ab) in Bezug auf die Dependency Rule]*

Die Dependency Rule besagt, dass die Abhängigkeiten immer von außen nach innen verlaufen. Aufgerufen werden können diese Klassen aus beiden Richtungen der Abstraktion.

### **Positiv-Beispiel: Dependency Rule**

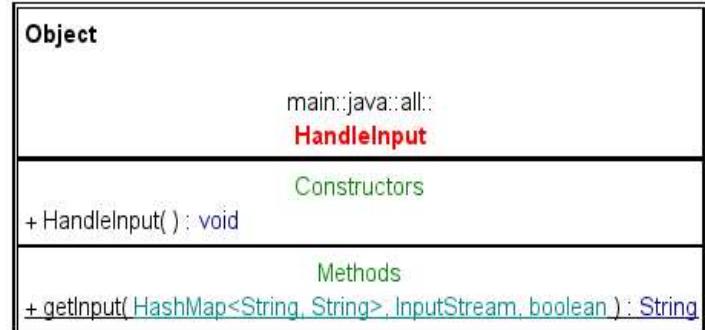
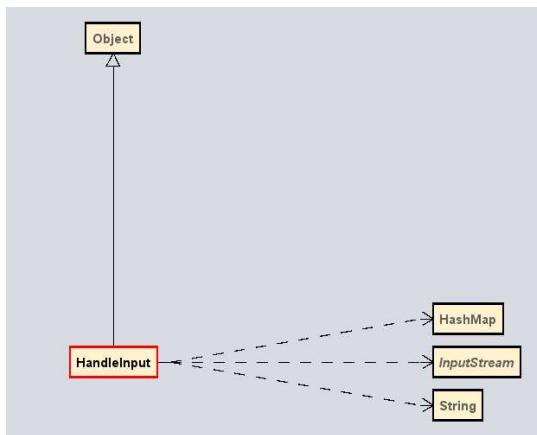
Das Interface „Questions“ ist Teil des Domaincodes. Dieses ist in den Klassen die die genauen Fragen der verschiedenen Kategorien darstellen implementiert. Von dieser Klasse hängt die Klasse „ParseQuestions“ ab.





### Negativ-Beispiel: Dependency Rule

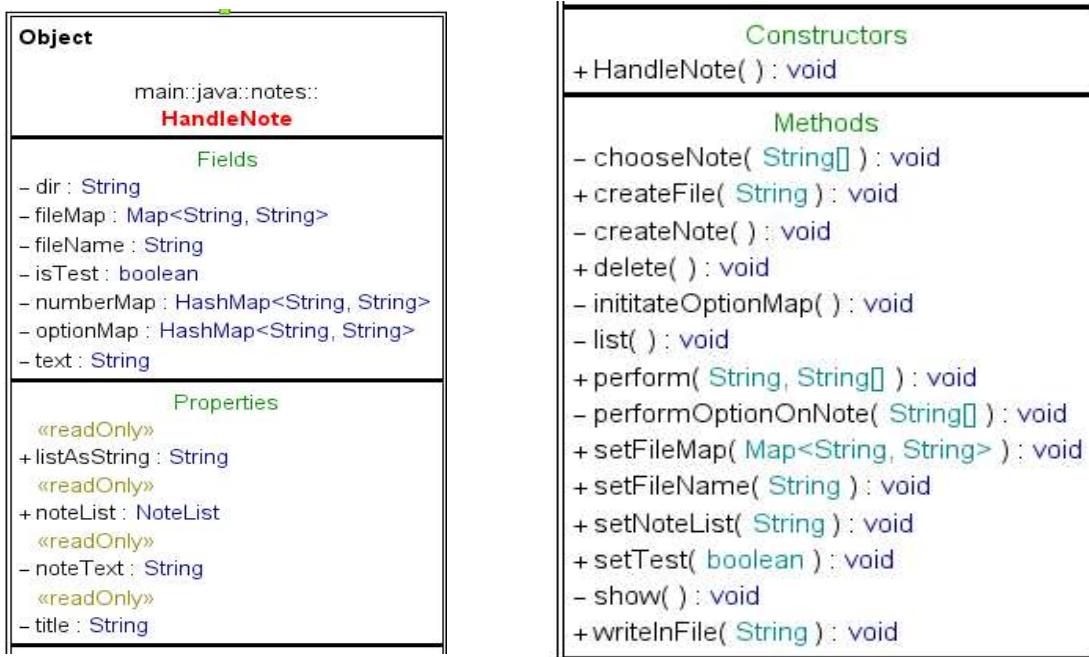
Die Klasse HandleInput, da diese die Information benötigt, für welche Map eine Eingabe eingeholt werden soll. Daher hängt die Klasse obwohl sie außen in der Pluginschicht liegt, von den Klassen im Domaincode ab. Diese geben der Klasse vor, welche Eingaben valide Eingaben sind und welche Zuordnung sie entsprechend zurückzugeben hat.



### Analyse der Schichten

[jeweils 1 Klasse zu 2 unterschiedlichen Schichten der Clean-Architecture: jeweils UML der Klasse (ggf. auch zusammenspielenden Klassen), Beschreibung der Aufgabe, Einordnung mit Begründung in die Clean-Architecture]

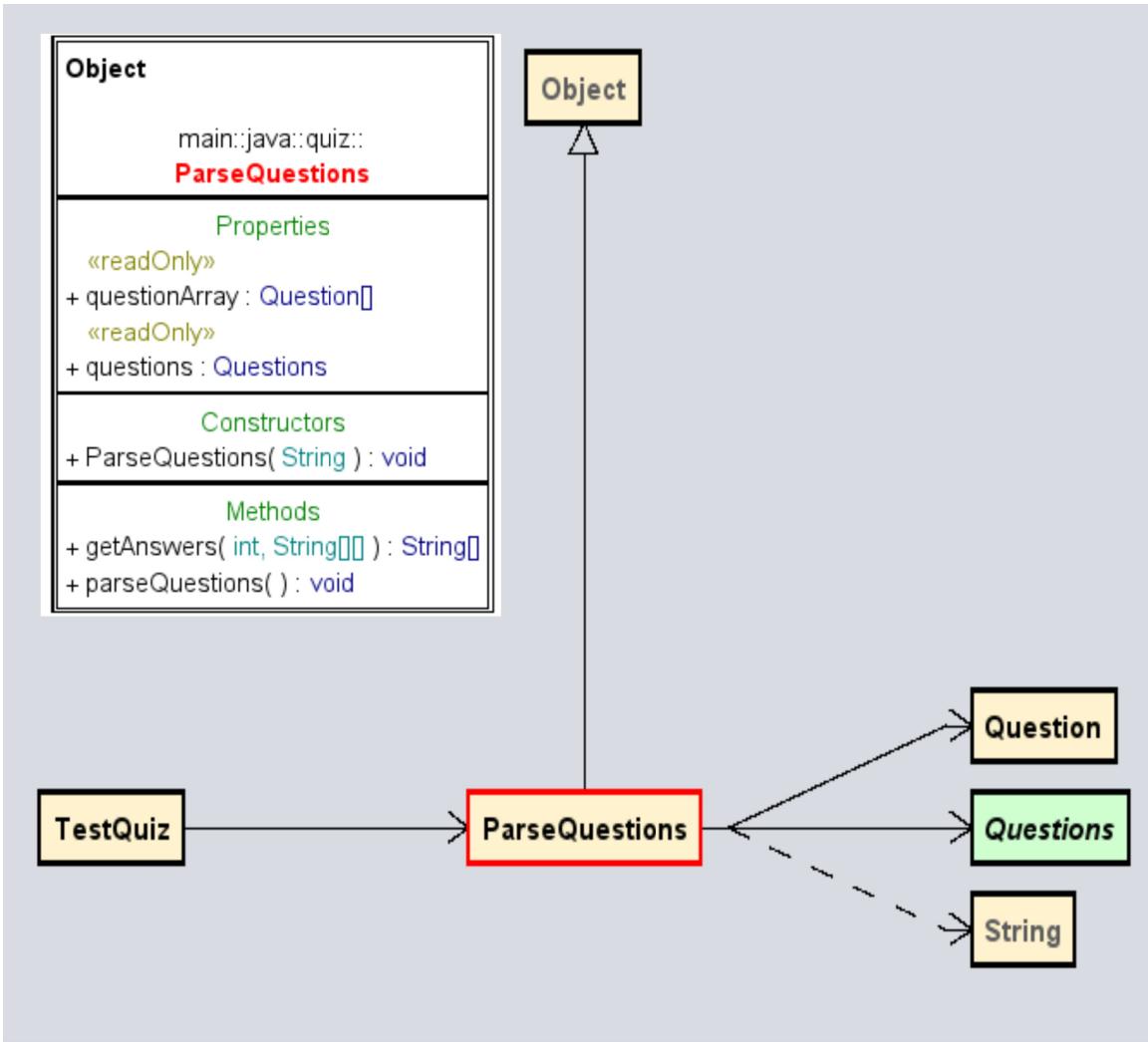
## Schicht: Application Code



Die Klasse „HandleNote“ beinhaltet die Methoden zur Handhabung einer Notiz. Daher ist diese der Application Schicht zuzuordnen. Aufgerufen wird diese Klasse aus der Klasse „NotesStartSite“, sollte es sich bei der Auswahl der Option um eine valide Aufgabe handeln. Die Klasse „HandleNote“ kümmert sich um das Erstellen einer Notiz, Auswählen einer Notiz, Löschen einer Notiz, Auflisten der Notizen, Erstellen einer Notiz und das Anzeigen einer bestimmten Notiz.

## Schicht: Adapters

Die Klasse „ParseQuestions“ dient wie der Name bereits andeutet als Umwandler der Fragen in das Question Objekt. Daher gehört diese in die Schicht der Adapter. Deren Aufgabe ist das Umwandeln der Fragen, welche in einem zweidimensionalen Array gespeichert sind, in das Question-Objekt. Dieses Objekt beinhaltet eine Frage, ein Array mit den Antwortmöglichkeiten und die richtige Antwort.

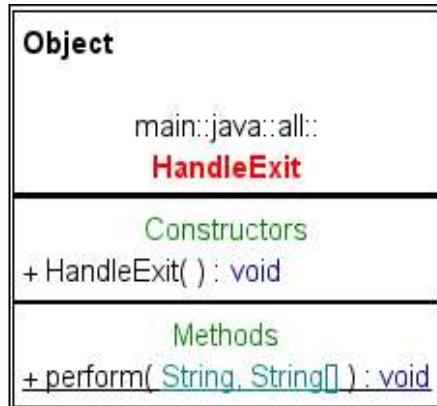


## Kapitel 3: SOLID

### Analyse Single-Responsibility-Principle (SRP)

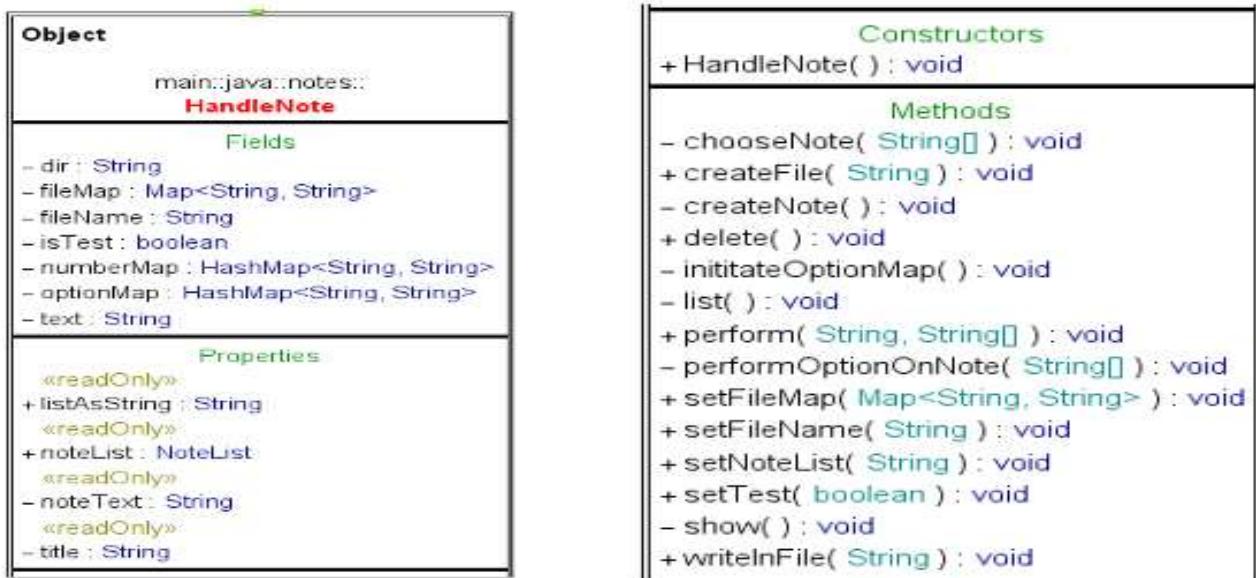
[jeweils eine Klasse als positives und negatives Beispiel für SRP; jeweils UML der Klasse und Beschreibung der Aufgabe bzw. der Aufgaben und möglicher Lösungsweg des Negativ-Beispiels (inkl. UML)]

## Positiv-Beispiel



Die Klasse HandleExit ist lediglich dafür da sich um den Ausstieg aus dem aktuellen Programmzweig zu kümmern. Beziehungsweise das Programm komplett zu beenden. Daher ändert sich diese Klasse nur, wenn sich das Verhalten bei dem auf den Start zurück gehen ändert. Ist die Option zurück zum Start aktiviert, ruft das Programm die Main-Klasse des Programms auf.

## Negativ-Beispiel

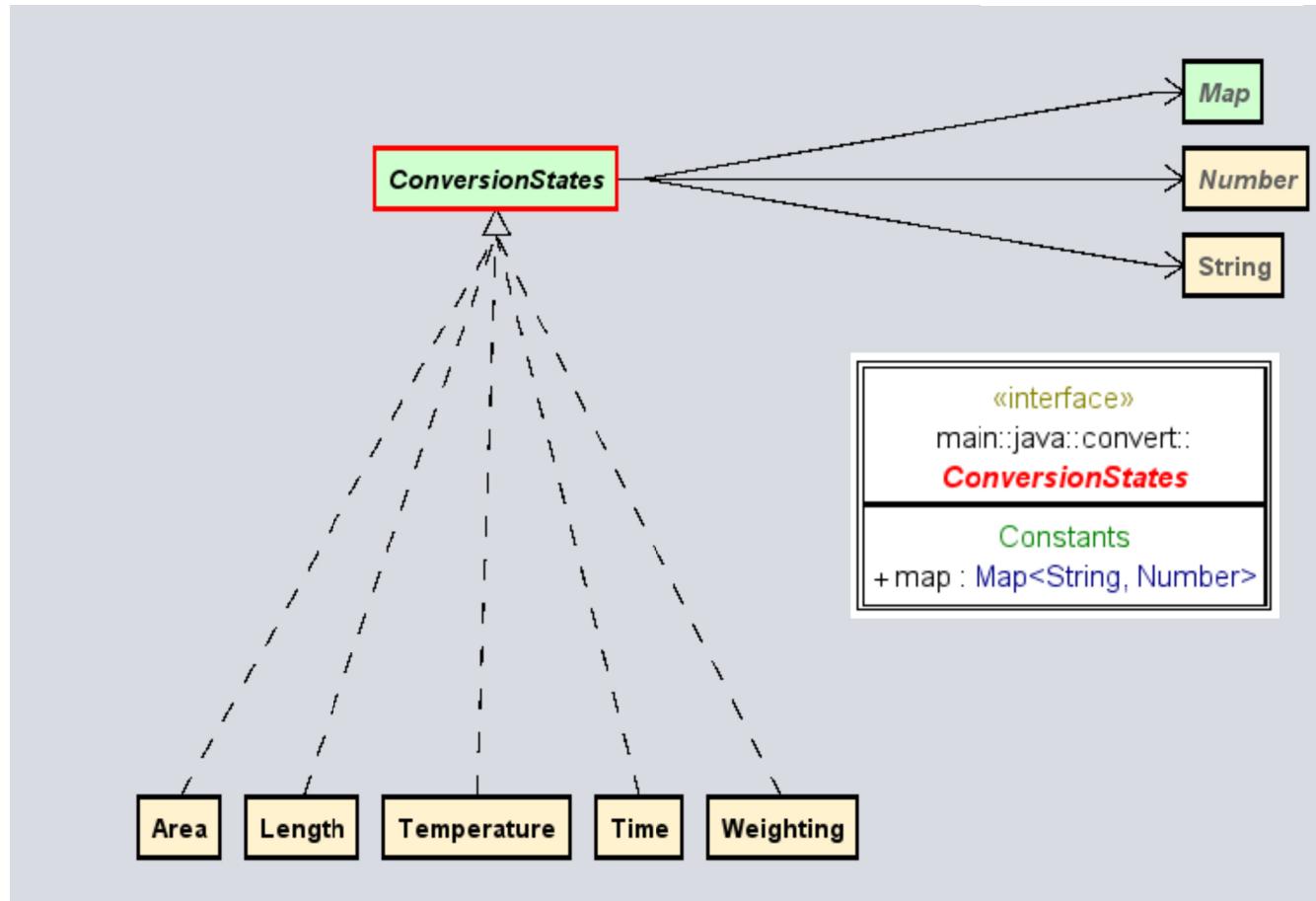


Die Klasse HandleNote behandelt alle Optionen, die mit einer Notiz ausgeführt werden können. Dies sind das Auswählen einer Notiz, das Erstellen einer Notiz, das Löschen einer Notiz, das Auflisten aller Notizen und das Anzeigen einer Notiz. Die mögliche Lösung für das Problem ist die Optionen in jeweils eine eigene Klasse auszulagern.

## Analyse Open-Closed-Principle (OCP)

[jeweils eine Klasse als positives und negatives Beispiel für OCP; jeweils UML der Klasse und Analyse mit Begründung, warum das OCP erfüllt/nicht erfüllt wurde – falls erfüllt: warum hier sinnvoll/welches Problem gab es? Falls nicht erfüllt: wie könnte man es lösen (inkl. UML)?]

## Positiv-Beispiel

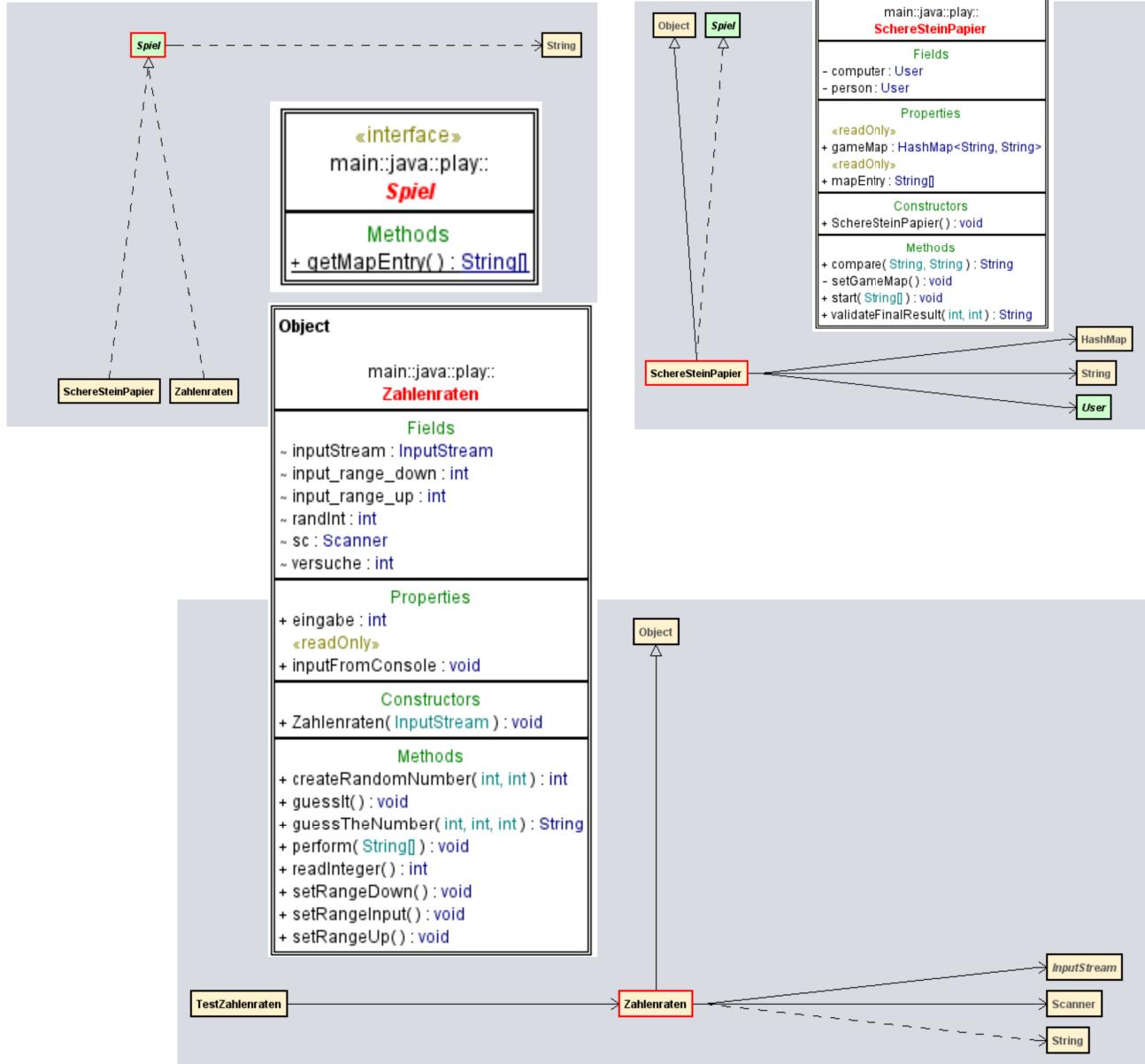


Das Interface **ConversionStates** abstrahiert die Maps der Umrechnungsfaktoren der verschiedenen Einheiten. Dies wurde eingeführt, da bei der Instanziierung der Umrechnungswerte die Map gespeichert werden musste und damit hier nicht für jeden Wert eine eigene Methode geschrieben werden musste. Dies wird durch eine weitere Umstrukturierung nicht benötigt, da direkt die Map zurückgegeben wird. Sollten weitere Einheiten umgerechnet werden wollen, können diese einfach als neue Klasse hinzugefügt werden.

## Negativ-Beispiel



Bei neuen Spielen hinzufügen existiert kein Interface. Beim Hinzufügen eines neuen Spiels muss die Klasse SpieleMain händisch aktualisiert werden. Es existiert kein Spiele Interface, über welches mittels einer Schleife die Klassen der Spiele hinzugefügt werden können. Die Struktur der Klassen sieht dann wie folgt aus:



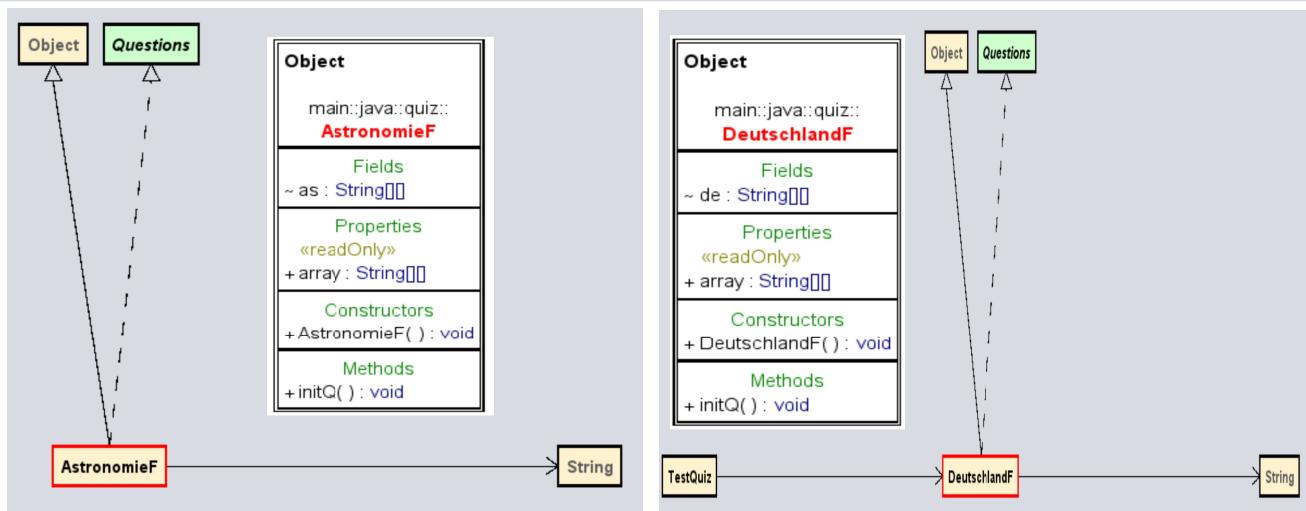
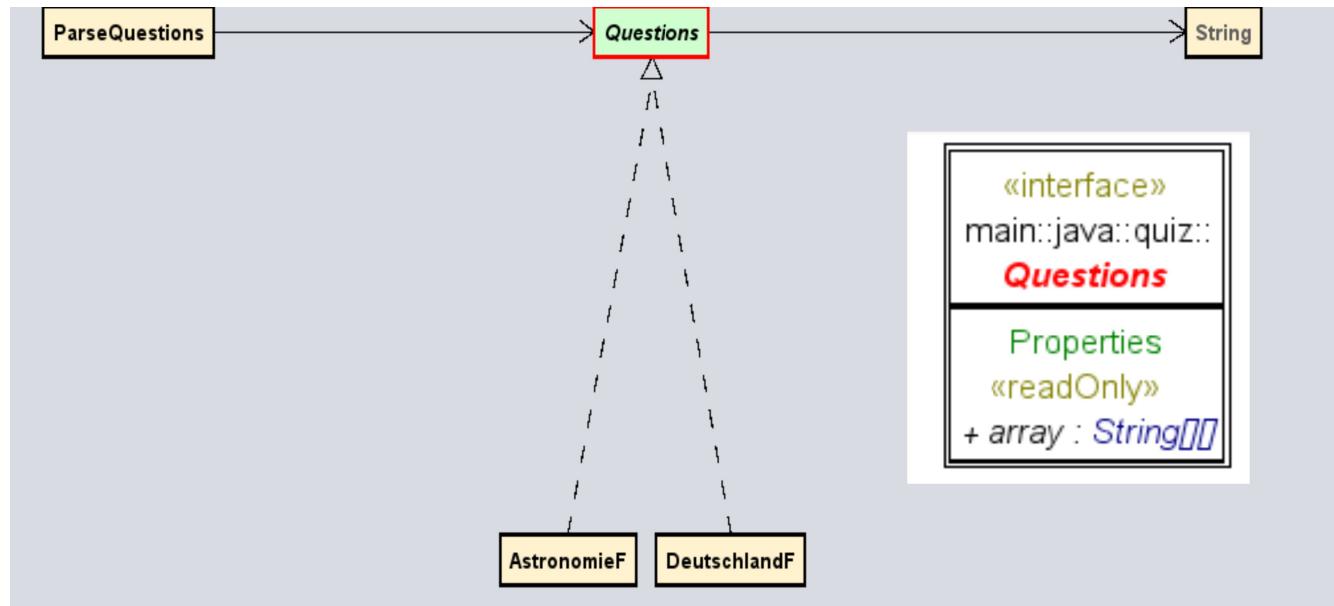
## Analyse Liskov-Substitution- (LSP), Interface-Segregation- (ISP), Dependency-Inversion-Principle (DIP)

[jeweils eine Klasse als positives und negatives Beispiel für entweder LSP oder ISP oder DIP); jeweils UML der Klasse und Begründung, warum man hier das Prinzip erfüllt/nicht erfüllt wird]

[Anm.: es darf nur ein Prinzip ausgewählt werden; es darf NICHT z.B. ein positives Beispiel für LSP und ein negatives Beispiel für ISP genommen werden]

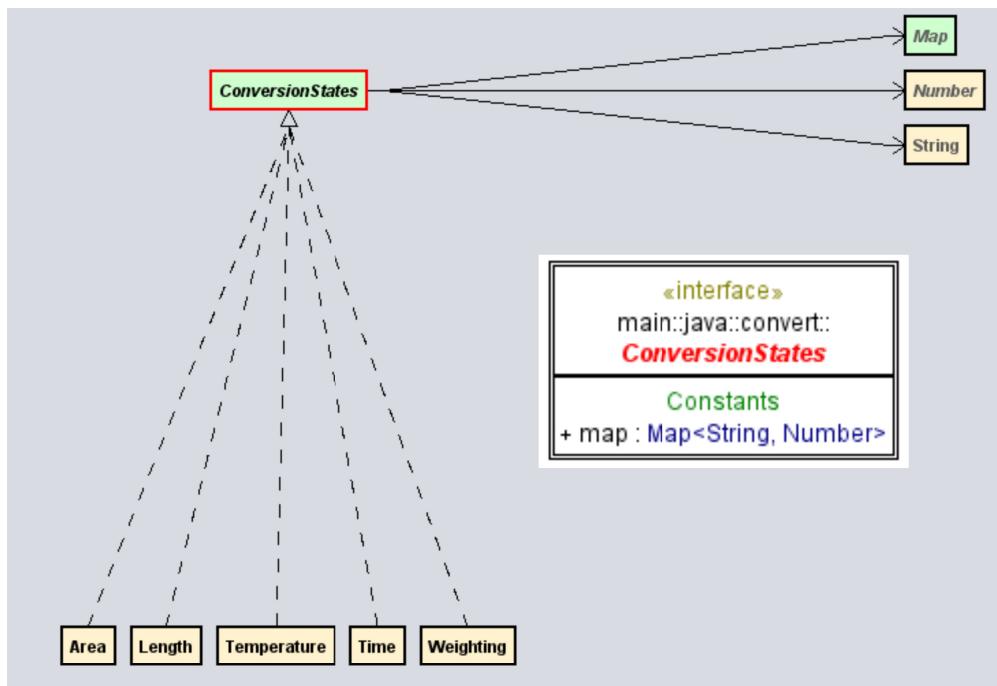
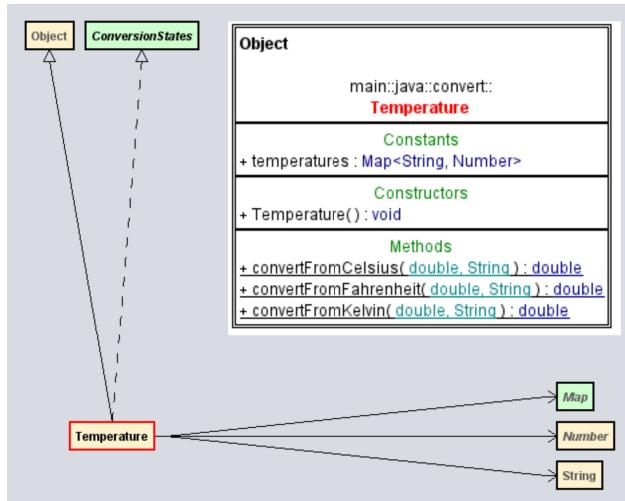
## Positiv-Beispiel

LSP: Das Interface Questions kann von jeder der dieses aktuell implementierenden Klasse ersetzt werden. Alle verhalten sich am Ende gleich, sie beinhalten ein Array der Fragen und beinhalten eine Methode dieses auszulesen.



## Negativ-Beispiel

Das Interface ConversionStates kann nicht durch alle dieses Interface implementierenden Klassen ersetzt werden. Dies liegt daran, dass die Formeln zum Umrechnen der Temperatur spezielle Formeln sind. Und daher die Klasse Temperature weitere Methoden implementiert, die die Umrechnung durchführen.



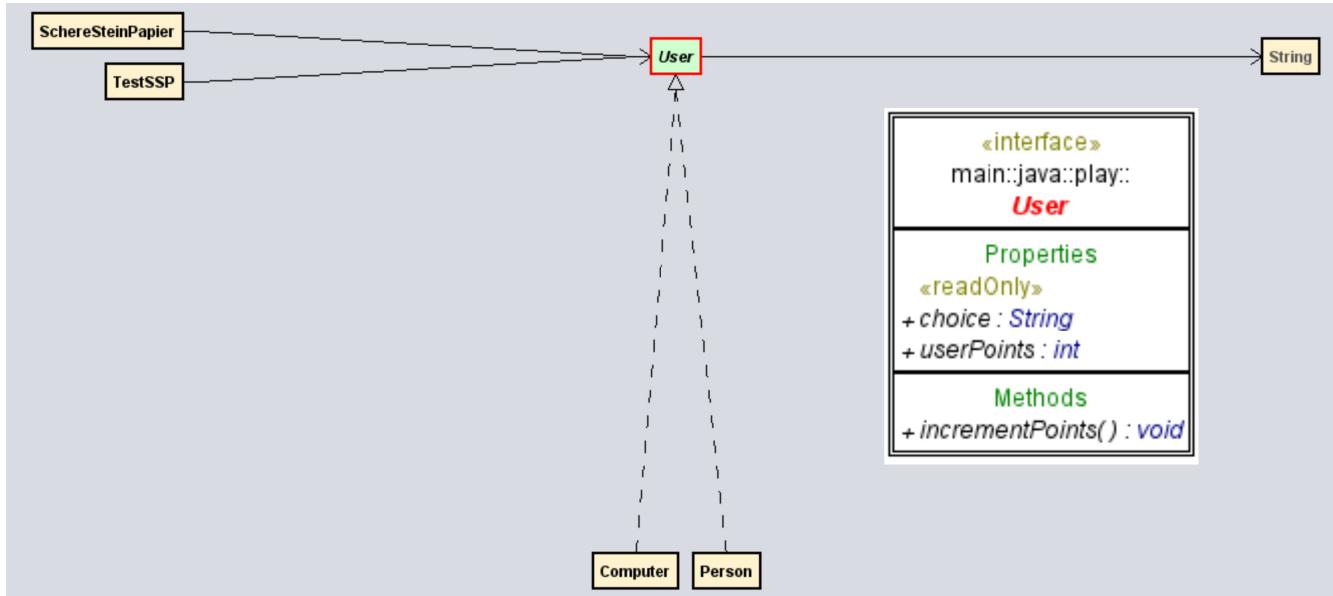
## Kapitel 4: Weitere Prinzipien

### Analyse GRASP: Geringe Kopplung

[jeweils eine bis jetzt noch nicht behandelte Klasse als positives und negatives Beispiel geringer Kopplung; jeweils UML Diagramm mit zusammenspielenden Klassen, Aufgabenbeschreibung und Begründung für die Umsetzung der geringen Kopplung bzw. Beschreibung, wie die Kopplung aufgelöst werden kann]

## Positiv-Beispiel

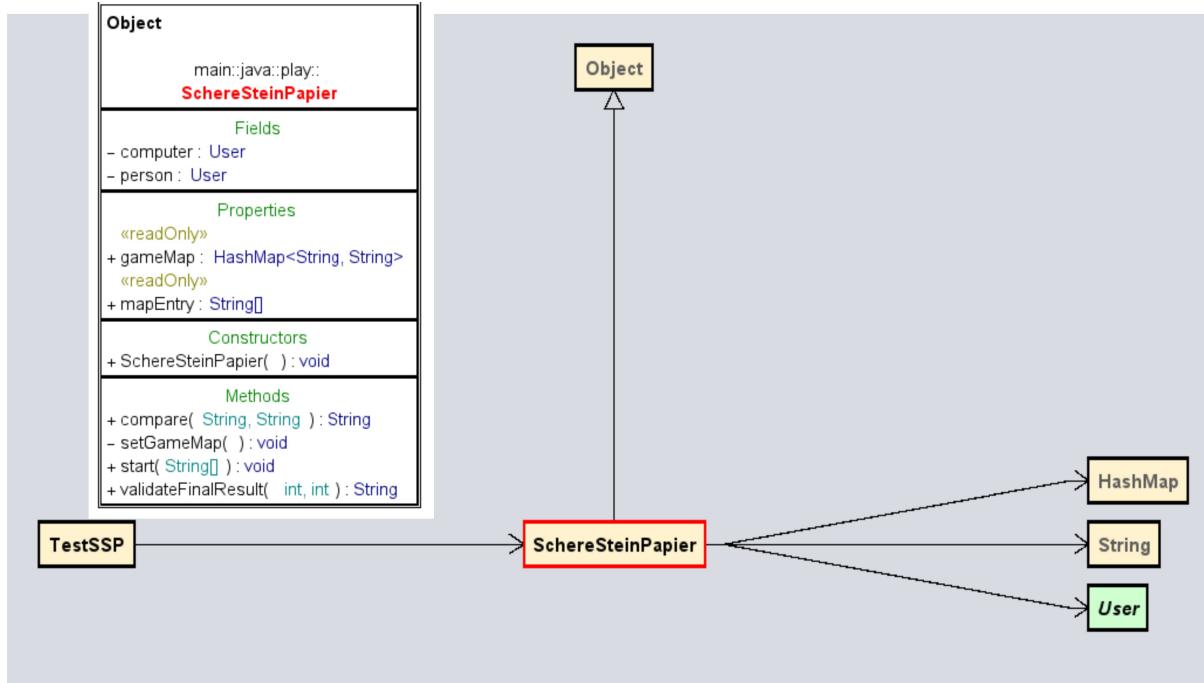
Das Interface User, welches in verschiedenen Formen implementiert werden kann. In diesem Beispiel ist dies die Methode getChoice. Diese generiert für den Computer automatisch eine Eingabe bzw. eine Entscheidung. Bei der Klasse Person ist das Abfragen einer Entscheidung über die Console einließt und überprüft, dass es sich um eine valide Eingabe handelt.



## Negativ-Beispiel

Die Startmethode in der Klasse SchereSteinPapier beinhaltet viel Logik und Programmcode in einer Methode. Daher liegt hier eine starke Kopplung vor. Als Beispiel ist hier der Programmcode eingefügt.

An dieser Stelle sehe ich keine Möglichkeit die Kopplung komplett aufzulösen. Eine leichte Verbesserung wäre das ausgliedern der Funktionalität des Durchführens einer Spielrunde in eine eigene Methode.



```

public void start(String[] args) {
    System.out.println("Willkommen bei Schere Stein Papier!\n" + "Wir spielen Best-Of-Three, viel Glueck!\n");
    String userInput;
    boolean aktiv = true;

    while (aktiv) {
        int counterRounds = 0;

        while (this.computer.getUserPoints() < 2 && this.person.getUserPoints() < 2) {
            counterRounds++;
            String pcChoice = this.computer.getChoice();
            System.out.println(counterRounds + ". Runde: Computer hat sich entschieden.\n"
                + "Was nimmst du? Eingabe in Kleinbuchstaben:\n");
            userInput = this.person.getChoice();
            System.out.println("userinput: " + userInput + "\n-----");

            System.out.println("user: " + userInput + "\nComputer: " + pcChoice + "\n-----");
            String erg = compare(pcChoice, userInput);

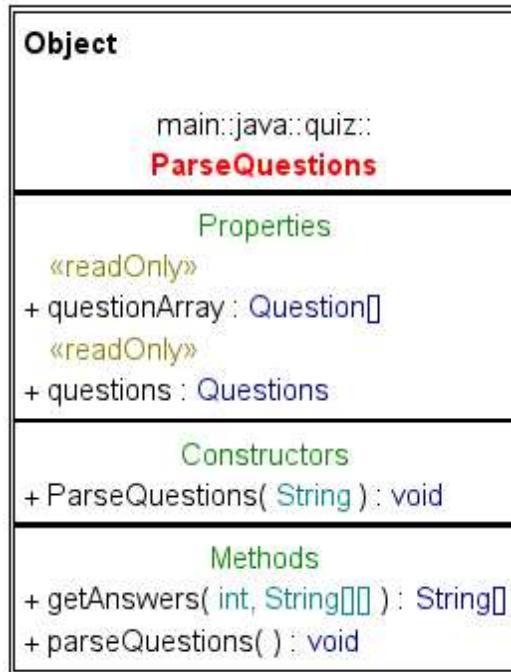
            System.out.println("pc: " + this.computer.getUserPoints() + " " + "user: " + this.person.getUserPoints());
        }
        System.out.println(validateFinalResult(this.person.getUserPoints(), this.computer.getUserPoints()));
        SpieleMain.main(args);
    }
}

```

## Analyse GRASP: Hohe Kohäsion

[eine Klasse als positives Beispiel hoher Kohäsion; UML Diagramm und Begründung, warum die Kohäsion hoch ist]

In der Klasse ParseQuestions erfüllt jede Methode lediglich eine Aufgabe. Daher ist hier eine hohe Modularität vorhanden und somit die Kohäsion hoch.



## Don't Repeat Yourself (DRY)

[ein Commit angeben, bei dem duplizierter Code/duplizierte Logik aufgelöst wurde; Code-Beispiele (vorher/nachher); begründen und Auswirkung beschreiben]

Commit nr: 772f548

Davor an jeder Stelle, wo eine Eingabe stattfindet geprüft, ob die Eingabe in den Optionen vorhanden ist und ggf nach einer erneuten Eingabe fragen. Danach Aufruf der Methode und bekommt den String aus der Map zurück, also den internen Wert des Programms. Die Validierung der Eingabe wird somit an eine eigene Klasse ausgelagert, die an jeder Stelle verwendet werden kann. Es ist lediglich eine Map notwendig, die die validen Eingaben und ihre zugehörigen Funktionen beinhaltet.

Der genaue Commit und somit die Änderungen sind im Anhang an das Dokument angehängt, da dieser hier nicht sinnvoll untergebracht werden konnte.

# Kapitel 5: Unit Tests

## 10 Unit Tests

[Nennung von 10 Unit-Tests und Beschreibung, was getestet wird]

Unit Test	Beschreibung
TestZahlenraten#testZahlenratenCreateRandomNumber	In der Methode createRandomNumber im Spiel Zahlenraten wird eine zufällige Zahl ausgewählt, die der User erraten soll. Für das Erzeugen dieser Zufallszahl wird eine untere sowie obere Grenze vom User angegeben. Um zu gewährleisten, dass die Zufallszahl der oberen und unteren Schranke nicht unter- oder überschreitet, wird mit diesem Test geprüft, ob die erzeugte Zufallszahl im richtigen Zahlenbereich liegen. Im Test werden statische Werte für die obere und untere Schranke sowie für die Zufallszahl angegeben, um somit die Methode zu testen.
TestSSP#testCompareDraw	Im Spiel Schere Stein Papier gibt es drei verschiedene Möglichkeiten, eine Runde zu beenden: unentschieden, gewonnen oder verloren. Dieser Test prüft das Ergebnis, wenn ein Unentschieden erzielt wurde. Für die anderen zwei Optionen gibt es jeweils auch eine Methode. Die Methode compare bekommt als Parameter zwei Strings (hier wieder statisch), die zufällige Auswahl des Computers und die Eingabe des Users. Die Methode kann als Rückgabe die Werte „0“ für verloren, „1“ für unentschieden und „2“ für gewonnen annehmen. Im Test testCompareDraw (Prüfung auf unentschieden), wird geprüft, ob die Rückgabe der Methode eine „1“ liefert, da die statischen Parameter beide gleich sind. Bei den anderen „compare“-Tests wird dies in der selben Weise geprüft, nur mit entsprechend unterschiedlichen Parametern, für das Gewinnen oder Verlieren. „1“ liefert, da die statischen Parameter beide gleich sind. Bei den anderen „compare“-Tests wird dies in der selben Weise geprüft, nur mit entsprechend unterschiedlichen Parametern, für das Gewinnen oder Verlieren.
TestEntrace#testHandleInputFalseInput	Die Methode getInput der Klasse HandleInput prüft die Eingaben auf Korrektheit bezüglich der in einer Liste gespeicherten akzeptierten Eingaben. Mit dieser Testmethode testHandleInputFalseInput soll geprüft und bestätigt werden, dass die Methode erfolgreich „false input“ ausgibt, wenn die Eingabe nicht der erwarteten Möglichkeiten entspricht. Es werden hier die Eingaben „1“ für Spielen, „2“ für Quiz, „3“ für Notizen, „4“ für Umrechnen und „X“ für Exit erwartet. Im Test wird der Eingabestring auf „test“ gesetzt. Zusätzlich dazu wird mittels der Klasse ByteArrayInputStream (Java.io) den String in Bytes umgerechnet, damit diese als Parameter übergeben werden können, um einen „echten“ User-Input zu simulieren.

	Dasselbe wird ebenso mit der Methode TestEntrace#testHandleInputRightInput durchgeführt, jedoch mit dem Ziel, dass die Methode die Eingabe als gültig erkennt und somit eine richtige Rückgabe erzeugt.
TestAreaConversation#testQmToQkm	In dieser Methode wird geprüft, ob die Umrechnung von Quadratmetern in Quadratkilometern stimmt. Es wird in der Testmethode mittels der Methode setUnit die Einheit bestimmt, die als Quelleingabe fungiert. Danach wird mittels der Methode setInputNumber die Zahl eingegeben, die in eine anderen Einheit umgerechnet werden soll. Hiernach setzt man schließlich mit der Methode setGoalUnit die gewünschte Einheit, die am Ende das Ergebnis darstellen soll. In der Testmethode assertEquals muss in diesem Fall ein dritter Parameter hinzugefügt werden, da es bei dem Vergleich von zwei Double-Werten aufgrund von Rundungsungenauigkeiten zu einem unterschiedlichen Ergebnis kommen kann, obwohl dasselbe Ergebnis rauskommen sollte.
TestNotes#testDeleteNote	In dieser Testmethode wird überprüft, ob eine Notizdatei nach dem Löschen noch existiert oder nicht, d.h. ob sie erfolgreich gelöscht worden ist oder nicht. Im Test werden zuerst Dateien erzeugt mittels der Methode createFiles(). Als nächstes wird dem zuvor erzeugten Objekt der Klasse HandleNote einen Dateinamen zugeordnet und anschließend mit der Methode delete() diese Datei wieder gelöscht. Es wird nun eine „Sammlung“ von Objekten angelegt mittels dem Set-Interface „response“, das alle Dateien enthält, die aktuell im Verzeichnis liegen. Um nun zu prüfen, ob die Datei erfolgreich gelöscht worden ist und somit nicht mehr existiert, wird die Testmethode assertFalse aufgerufen. Diese enthält als Parameter den Aufruf response.contains(this.filename). Dieser Aufruf prüft, ob der angegebene Dateiname im Set-Interface „response“ enthalten ist. Wenn die Datei erfolgreich gelöscht worden ist, kommt hier der Wert false raus, was auch das Wunschergebnis der Methode assertFalse ist.
TestQuiz#testValidateAnswerRightAnswer	Beim Quiz hat der Benutzer vier Antwortmöglichkeiten zu jeder Frage, die ihm gestellt wird. In diesem Test wird beispielhaft auf die erste Frage des Deutschland-Quiz zugegriffen. Es soll hier die Methode Quiz#validateAnswer getestet werden. Diese bekommt als Parameter der Input-String vom Benutzer und zum anderen der String mit der richtigen Antwort. Diese Methode soll bei Übereinstimmung dieser beiden Strings „Richtige Antwort“ und bei Nichtübereinstimmung „Falsche Antwort“ zurückgeben. In diesem Fall wird die Antwort ‚c‘ als Input eingegeben und mit der richtigen Antwort verglichen, sodass „richtige Antwort“ rauskommen muss.
TestNotes#testShowNote	In diesem Test wird geprüft, ob die Methode showNote funktioniert. Diese ist dafür zuständig, dass der Inhalt einer Notizdatei (richtig) angezeigt wird. Für diesen Test wird eine Datei mit der Methode createFile erstellt. Anschließend wird mit der Methode writeInFile in die Datei geschrieben. Es wird dort ein Text eingetragen, der mit der

	Methode getText innerhalb der Test-Klasse bestimmt wird. Anschließend wird mit assertEquals geprüft, ob der Inhalt der Datei, welcher mittels „this.hnote.getNoteList().readfile(this.fileName)“ ermittelt werden kann, mit dem Text der Methode getText übereinstimmt. Wenn ja, wurde die Datei richtig beschrieben und schlussendlich korrekt angezeigt.
TestNotes#testListNotes	In dieser Testmethode wird getestet, ob das Anzeigen lassen der Notizen mittels der Funktion listNotes funktioniert. Hier wird zunächst eine Map erstellt (setFileMap), die alle Dateinamen im aktuellen Verzeichnis enthält. Danach wird die Methode getListAsString ausgeführt, die die Namen der enthaltenen Dateien in einem String (response) abspeichert. Als Nächstes wird jeder existierende Dateiname (definiert durch die Liste „fileNames“) mit dem String response auf Teilmenge geprüft, mittels einer selbst implementierten „assertContains“ Testmethode. Diese nimmt als Parameter den String response mit all den verfügbaren Dateinamen und den Namen der aktuellen Datei. Die Methode assertContains überprüft mittels der Testmethode assertTrue ob der zweite Parameter (aktuelle Datei) ein Substring vom String response ist. Wenn dem so ist und die Methode true zurückgibt, so wurde die Methode listNotes erfolgreich getestet.
TestQuiz#testGetAnswers	Die Methode getAnswers in der Klasse ParseQuestion ist dafür verantwortlich, aus dem 2 dimensionalen String-Array die möglichen Antworten je Frage rauszufiltern, damit sie später einfacher in der Konsole ausgegeben werden können. In der Testmethode wird zuerst die Methode setAnswers aufgerufen. Diese führt zuerst die initQ Methode der Klasse DeutschlandF auf, die die Fragen, Antwortmöglichkeiten sowie der Buchstabe der richtigen Antwort initialisiert. In der Test-Klasse wird ein einfaches Array (this.answers) der Größe 4 erstellt, in dem die vier Antwortmöglichkeiten der ersten Frage drin stehen. Um nun zu prüfen, ob die Methode getAnswers richtig funktioniert, werden zwei Arrays mittels der Methode assertEquals verglichen. Der erste Array ist das zuvor erwähnte this.answer, welcher bereits den richtigen Inhalt besitzt. Nun wird als zweiter Parameter die Methode parseQ.getAnswer aufgerufen, welcher ein Array zurückgibt. Als Parameter dieser Methode wird der Index 0 (für die erste Frage) und ein vollständiges Array mit allen Fragen und Antworten drin. Wird dieser Test true, spaltet die Methode die Fragen-Blocks korrekt in die jeweiligen Antwortmöglichkeiten auf.
TestTemperatureConversion#testCelsiusToFahrenheit	Diese Testmethode prüft, ob das Umrechnen von der Temperatureinheit Celsius zu der Temperatureinheit Fahrenheit stimmt. Es wird zunächst die Einheit mittels setGoalUnit angegeben, in die umgerechnet werden soll. Gefolgt wird dies durch den Aufruf von setInputNumber. Diese Methode setzt den Celsius-Wert, welcher umgerechnet werden soll. Mittels dem Methodenaufruf setUnit wird die Einheit ausgewählt, die vom Benutzer eingegeben

wird. Danach wird die Method perform aufgerufen, die die Umrechnung umsetzt und den Wert des Ergebnisses in die Variable output speichert. Um in diesem Test nun zu prüfen, ob die Umrechnung richtig durchgeführt wurde, wird mittels assertEquals die beiden Werte 123.17 (Ergebnis in Fahrenheit) und die Variable output verglichen. Zusätzlich befindet sich hier ein dritter Parameter, der die Varainz angibt; also wie viel darf das Ergebnis vom angegebenen, ersten Parameter abweichen. Denn das Rechnen mit Kommazahlen in der Programmierung kann durchaus zu Rundungsfehlern kommen.

## ATRIP: Automatic

*[Begründung/Erläuterung, wie ‘Automatic’ realisiert wurde]*

Es wurde das JAVA-Testframework JUnit mit der Version 4 verwendet. Für die Testklassen der jeweiligen Programm-Kategorien wurde ein separates Package erstellt. In Eclipse können die einzelnen Tests mit der Tastatur-Kombination ALT + Shift + X ausgeführt werden. Optional kann auch mit Rechtsklick das Starten des Tests manuell mit der Maus gestartet werden. Die entwickelten Tests laufen alle automatisch und enthalten keine zwischenzeitlichen Benutzereingaben. Tests, die Methoden mit „eigentlichen“ Benutzerdaten testen, haben entweder statische Eingaben bekommen oder wurden (auch statisch) durch die Klasse ByteArrayInputStream injiziert. Damit die Tests korrekt sind und sich selbstständig überprüfen können, wurden Assertions verwendet.

## ATRIP: Thorough

*[jeweils 1 positives und negatives Beispiel zu ‘Thorough’; jeweils Code-Beispiel, Analyse und Begründung, was professionell/nicht professionell ist]*

Ein positives Beispiel ist das Testen der Methode, die den User Input regelt und entscheidet, ob der empfangene Input legitim ist. Folgend wird die Methode dargestellt und erläutert.

```
public static String getInput(HashMap<String, String> map, InputStream systemIn) {
    Scanner sc = new Scanner(systemIn);
    String input = sc.nextLine().toUpperCase();
    while(!map.containsKey(input)) {
        if(systemIn instanceof BufferedInputStream) {
            System.out.println("Falsche Eingabe, versuch es erneut:");
            input = sc.nextLine().toUpperCase();
        }else {
            return "false input";
        }
    }
    return map.get(input);
}
```

Als Parameter werden eine Map und ein Input Stream mitgegeben. In der Map sind die möglichen Inputs, die der Benutzer eingeben kann, gespeichert. In dieser Methode wird geprüft, ob der Input des Users einem Eintrag der Map entspricht. Ist dies nicht der Fall, wird eine Fehlermeldung „Falsche Eingabe, versuch es erneut:“ ausgegeben. Ist das eingegebene Input in der Map enthalten, wird der dazugehörige Value des Keys zurückgegeben. Da es zu dieser Methode einen Test gibt, wurde eine if-Abfrage dazwischengeschaltet, um zu prüfen ob der InputStream eine Eingabe des Benutzers ist (BufferedReader) oder ob es eine gefälschte Eingabe vom Test ist. Sollte es ein Test sein, wird kein Scanner aufgerufen, da es nicht sein soll, in einem Test eine manuelle Eingabe miteinzubringen. Die Unterscheidung zwischen Test oder nicht-Test wird hier daher als professionell eingestuft.

Ein negatives Beispiel ist das Testen einer „typischen“ get-Methode. Hier wäre es die Methode `getEingabe()` in der Klasse Zahlenraten. Die Methode wird in der folgenden Grafik dargestellt.

```
public int getEingabe() {  
    return this.eingabe;  
}
```

Die Methode gibt den Wert der aktuellen Eingabe des Objekts zurück. In der Testklasse wird diese Methode getestet, indem mithilfe der Methode `setEingabe()` der Variable einen Wert zugewiesen wird. Mit einem `assertEquals([Wert], [Objekt].getEingabe())` kann auf Gleichheit geprüft werden. Sollte `assertEquals` wahr sein, funktioniert die Methode `getEingabe` korrekt. Als professionell kann dies nicht eingestuft werden, da diese Methoden trivial sind und auch von IDEs automatisch erzeugt werden lassen können.

## ATRIP: Professional

[jeweils 1 positives und negatives Beispiel zu ‘Professional’; jeweils Code-Beispiel, Analyse und Begründung, was professionell/nicht professionell ist]

Ein negatives Beispiel hierfür ist das Testen einer get-Methode. In der Klasse Zahlenraten kommen Getter und Setter vor für die Variable „eingabe“. In der Testklasse TestZahlenraten wird diese Methode getestet. Dies ist entspricht nicht der vollen Professionalität, da das Testen dieser Methode trivial und unnötig ist. Es kann zwar geprüft werden, dass sie richtig funktioniert, ist aber aus Sicht des Entwicklers nicht essentiell wichtig für die Tests sowie der allgemeinen Funktionalität.

Ein positives Beispiel ist in der Testklasse TestNotes, in der die Klasse Notes getestet wird. Hier werden extra Methoden innerhalb des Tests erstellt. Diese Methoden stellen an sich keinen Test da, werden jedoch trotzdem erstellt, um andere Tests einfacher und „realer“ ausführen zu können. Ein Beispiel ist hier die Methode „createfile()“:

```
private void createFile() {
    this.hnote.createFile(this.fileName);
    assertTrue(new File(dir.getPath() + File.separator + this.fileName).exists());
}
```

Diese Methode wird in zwei weiteren Methoden aufgerufen: in testCreateNote() und testShowNote(). Die häufigere Verwendung dieser Methode kann als professionell angesehen werden, da es die Tests im Allgemeinen übersichtlicher sowie verständlicher macht. Des Weiteren sind die einzelnen Methode, die getestet werden sollen, somit einfacher zu realisieren.

## Code Coverage

*[Code Coverage im Projekt analysieren und begründen]*

Die Code Coverage wurde mithilfe des Tools „EclEmma“ durchgeführt. Es wurde „Line Coverage“ verwendet. Hier geht es drum zu prüfen, wie viele und welche Zeilen in einem Test durchlaufen wurden.

Im nachfolgenden Bild ist zu sehen, dass die gemessene Code Coverage nur 33% beträgt. Das hat mehrere Gründe. Zum einen liegt es daran, dass das Tool EclEmma auch die Testklassen mit in das Code Coverage berücksichtigt. Da die Testklassen die Klassen sind, die die eigentlichen Tests durchführen und es somit keine Testabdeckung für diese Testklasse geben kann, wurde die Prozentzahl deutlich verringert. Um einen plausibleren Wert der Code Coverage zu enthalten, wurde die Anzahl der Test-Zeilen von der gesamten Zeilenanzahl entfernt. Im Anschluss wurde die neue Gesamtanzahl ins Verhältnis zu den getesteten Zeilen an Code gesetzt. Das neue Ergebnis der Code Coverage beträgt 48,37%.

Dieser immer noch relativ geringe Wert erklärt sich damit, dass in den Klassen in denen eine geringe Testabdeckung gibt, viele Konsolenausgabe für den User sind, die in den Tests außen vorgelassen wurden . In Anbetracht dessen, dass es keinen Sinn ergibt Konsolenausgaben in den Test miteinzubringen, da diese zu keinem Zeitpunkt beachtet werden, sind diese Codezeilen nicht von Tests abgedeckt. Beispiele hierfür sind Menüausgaben, die Ergebnisse von Auswertungen, Anfragen nach Eingaben sowie Hinweise für den Programmablauf. Zusätzlich dazu gibt es viele Verzweigungen im Projekt. Dazu zählen z.B. die Übergänge zwischen den einzelnen Komponenten, wie aus dem Hauptmenü in einzelne Untermenüs zu gelangen oder ins Hauptmenü wieder zurück zu kommen. Das erklärt die geringe Testabdeckung in Klassen, die nur für die erfolgreiche Verzweigung in die Projektkomponenten zuständig sind.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Fun&Learn	33,8 %	1.458	2.851	4.309
src	33,8 %	1.458	2.851	4.309
test.java.convert	0,0 %	0	495	495
main.java.quiz	54,7 %	556	461	1.017
AstronomieF.java	0,0 %	0	266	266
QuizJava	37,9 %	69	113	182
QuizMain.java	0,0 %	0	68	68
Question.java	57,1 %	12	9	21
ParseQuestions.java	94,3 %	83	5	88
DeutschlandF.java	100,0 %	392	0	392
main.java.play	32,4 %	214	446	660
Zahlenraten.java	27,6 %	72	189	261
SchereSteinPapier.java	32,5 %	83	172	255
SpieleMain.java	0,0 %	0	70	70
Person.java	71,4 %	20	8	28
Computer.java	84,8 %	39	7	46
test.java.play	0,0 %	0	355	355
main.java.notes	49,4 %	320	328	648
HandleNote.java	50,3 %	225	222	447
NotesStartSite.java	0,0 %	0	59	59
NoteList.java	66,9 %	95	47	142
test.java.notes	0,0 %	0	249	249
main.java.convert	63,8 %	342	194	536
Conversion.java	66,4 %	188	95	283
ConversionStartSite.java	0,0 %	0	75	75
Temperature.java	84,7 %	50	9	59
Area.java	89,7 %	26	3	29
ConversionStates.java	0,0 %	0	3	3
Length.java	89,7 %	26	3	29
Time.java	89,7 %	26	3	29
Weigthing.java	89,7 %	26	3	29
main.java.all	17,0 %	26	127	153
FunLearn.java	0,0 %	0	96	96
HandleExit.java	0,0 %	0	18	18
HandleInput.java	66,7 %	26	13	39
test.java.quiz	0,0 %	0	109	109
test.java.all	0,0 %	0	87	87

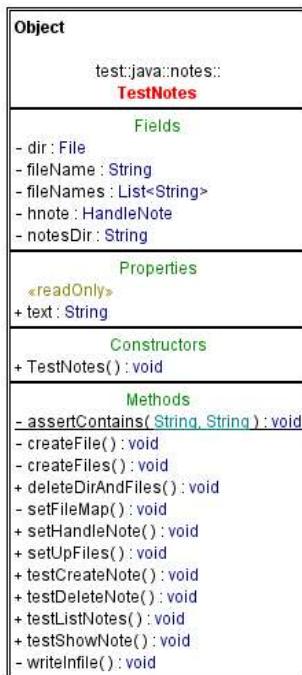
## Fakes und Mocks

[Analyse und Begründung des Einsatzes von 2 Fake/Mock-Objekten; zusätzlich jeweils UML Diagramm der Klasse]

In der Testklasse „TestNotes“ werden zwei Mock-Objekte verwendet, die geholfen haben, die benötigten Tests für die Klasse Notes durchzuführen. Das erste Mock-Objekt ist das Objekt von HandleNote. Dies ist das Hauptobjekt, das in diesem Test verwendet wird. Da es viele abhängige Methoden hat, kann es schwierig sein, alle seine Funktionalitäten im Test zu überprüfen. Ein Mock-Objekt kann erstellt werden, um die Abhängigkeiten von HandleNote zu isolieren und es einfacher zu machen, seine Funktionalität zu testen. Das Mock-Objekt für HandleNote wird verwendet um sicherzustellen, dass die Methoden zum Erstellen von Dateien, Schreiben von Text und Löschen von

Dateien ordnungsgemäß aufgerufen werden, ohne tatsächlich Dateien im Dateisystem zu erstellen, welche für die reale Anwendung wichtig wären.

Das zweite Mock-Objekt ist das Objekt von “File”. In diesem Code werden ein Verzeichnis und Dateien erstellt, gelesen und geschrieben. Die Verwendung von realen Dateien kann jedoch zu unerwünschten Nebeneffekten führen und auch schwieriger zu testen sein. Deshalb kann es sinnvoll sein, ein Mock-Objekt für die Datei zu erstellen, das die gleiche Schnittstelle wie das reale Objekt hat, aber keine tatsächlichen Dateioperationen ausführt. Das Mock-Objekt für File wird verwendet um sicherzustellen, dass die Methoden zum Lesen von Dateien oder zum Überprüfen der Existenz von Dateien ordnungsgemäß aufgerufen werden, ohne dass tatsächlich auf das Dateisystem real zugegriffen wird. Folgend das UML-Diagramm der Testklasse TestNotes.



## Kapitel 6: Domain Driven Design

### Ubiquitous Language

[4 Beispiele für die Ubiquitous Language; jeweils Bezeichnung, Bedeutung und kurze Begründung, warum es zur Ubiquitous Language gehört]

Bezeichnung	Bedeutung	Begründung
Umrechnung	Teil der Fähigkeiten des Programms und umrechnen von einer Einheit in eine andere	Bei Umrechnen denkt ein Entwickler zunächst an das Anpassen einer Datenstruktur an eine andere. Hier geht es aber um das Umrechnen verschiedener physikalischer Einheiten.
Notizen	Selbe Bedeutung wie das Wort Notizen in der normalen Sprache	Da es dem natürlichen Sprachgebrauch angehört, zählt es in diesem Projekt zur Ubiquitous Language.

Schere-Stein-Papier	Selbe Bedeutung wie das Da es dem natürlichen Sprachgebrauch angehört, Wort Notizen in der normalen Sprache zählt es in diesem Projekt zur Ubiquitous Language.
Quiz	Selbe Bedeutung wie das Da es dem natürlichen Sprachgebrauch angehört, Wort Notizen in der normalen Sprache zählt es in diesem Projekt zur Ubiquitous Language.

## Entities

*[UML, Beschreibung und Begründung des Einsatzes einer Entity; falls keine Entity vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]*

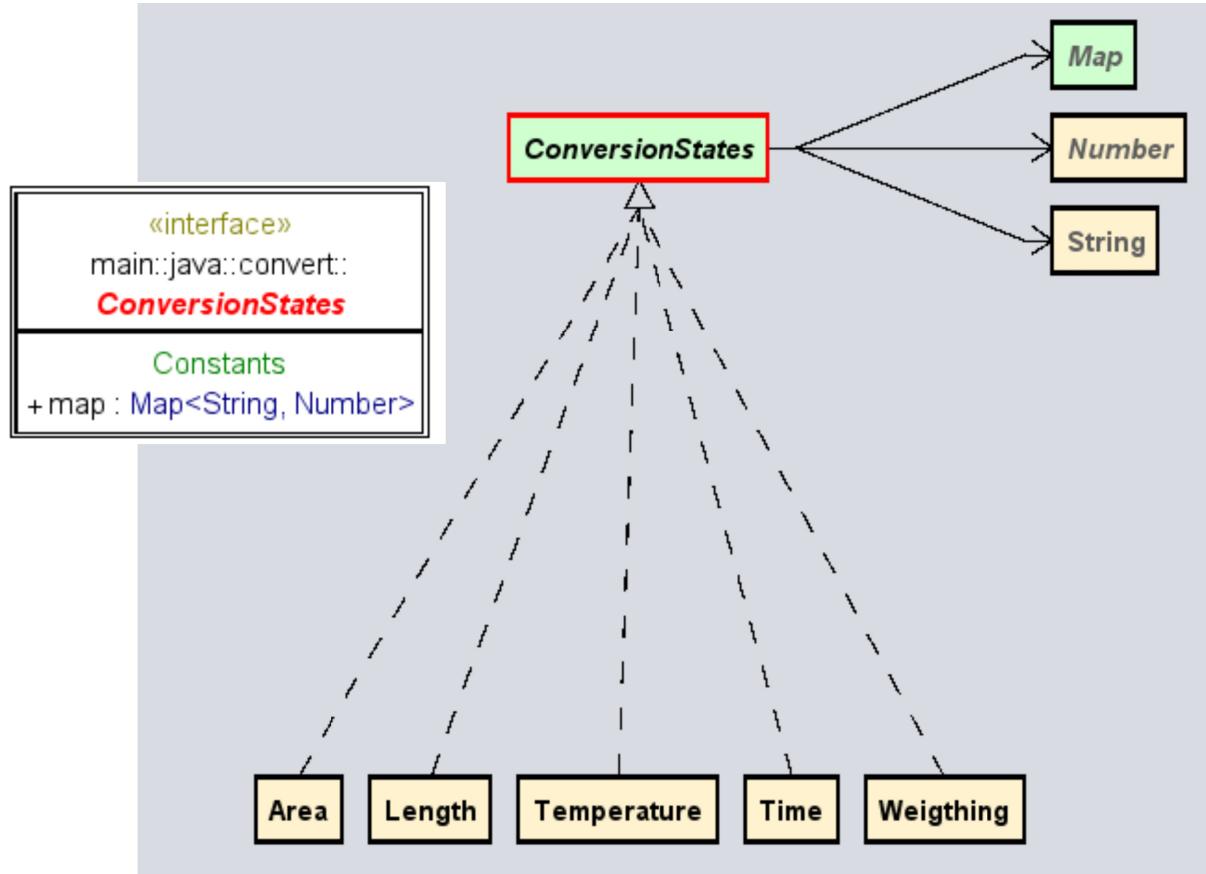
Entities haben eine eindeutige ID, da in dem vorliegenden Programm jeweils nur eine Instanz der verschiedenen Klassen gleichzeitig aktiv ist, existieren keine IDs. Es existieren Domainenregeln, die mit Objekten und deren Attributen abgebildet werden. Damit soll auch ohne eindeutige IDs sicher gestellt werden, dass die Objekte den Regeln der Domain entspricht. Die Objekte verändern sich im Laufe ihrer Existenz, durch die Existenz von Setter-Methoden und es gibt zu viele nicht genau definierbare ungültige Zustände, die nicht abgefangen werden können. Das Verhalten der Klassen ist zwar in Methoden, jedoch nicht in extra Klassen ausgelagert.

## Value Objects

*[UML, Beschreibung und Begründung des Einsatzes eines Value Objects; falls kein Value Object vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]*

Die Umrechnungstabellen der verschiedenen Einheiten könnten Value Objekte sein. Diese verändern sich prinzipiell nicht, bloß wenn weitere Maßeinheiten hinzugefügt werden. Das es allerdings während der Ausführung unveränderlich ist, ist damit sicher gestellt, dass es lediglich ein Getter gibt und die Variable private ist. Die Eigenschaften, die ein anderes Objekt darstellen sind einerseits die Maßeinheiten in der Map und andererseits die Namen der Klasse. Diese implementieren das Interface ConversionStates.

Als UML ist hier vertretend für alle Umrechnungsklassen das Interface der Umrechnung dargestellt.



## Repositories

[UML, Beschreibung und Begründung des Einsatzes eines Repositories; falls kein Repository vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

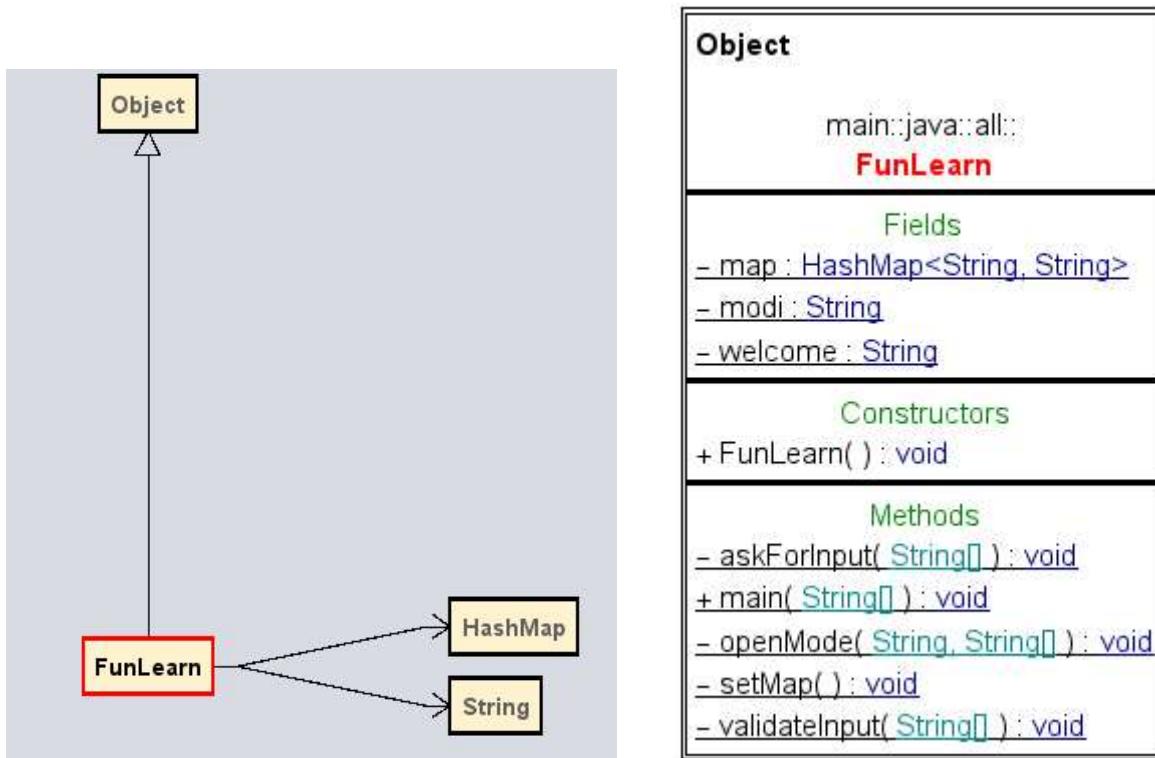
Das einzige, wo Zugriff auf den persistenten Speicher notwendig ist, sind die Notizen. Hierbei ist dem Domainen Code also der Logik, welche auszuführen ist, bekannt wie die Notizen im Speicher abgelegt werden sollen. Dies geschieht in Form von Textdateien auf die aus dem Code heraus direkt zugegriffen wird. Für diese geringen Umstände ist es überdimensioniert und sich wiederholend die Interaktion mit dem Speicher in eine extra Klasse auszulagern. Daher existiert kein Repository in einer eigenen Klasse und somit kann auch kein UML angegeben werden.

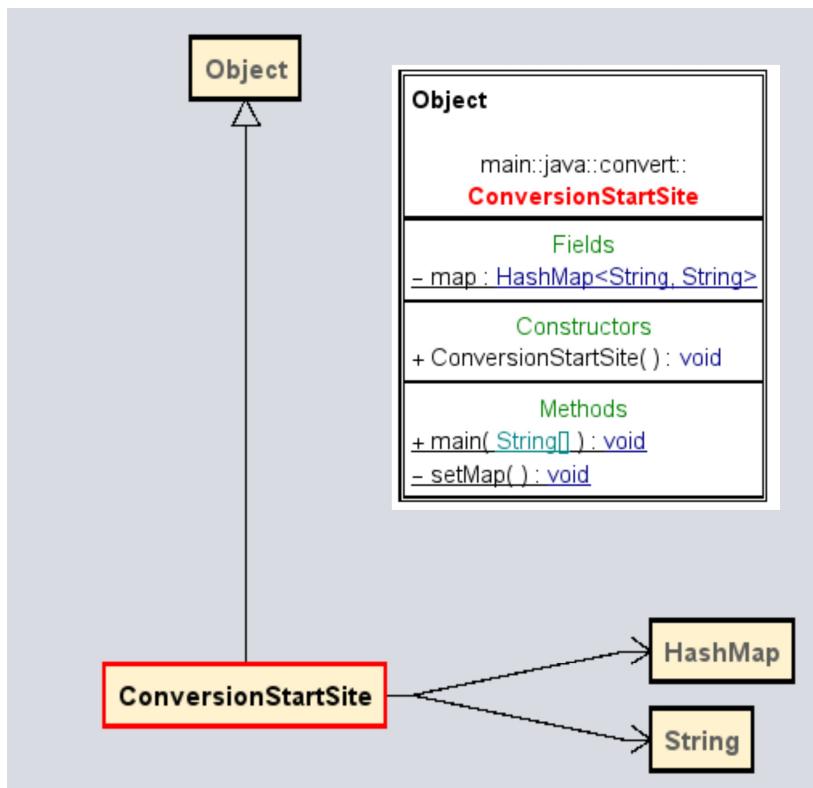
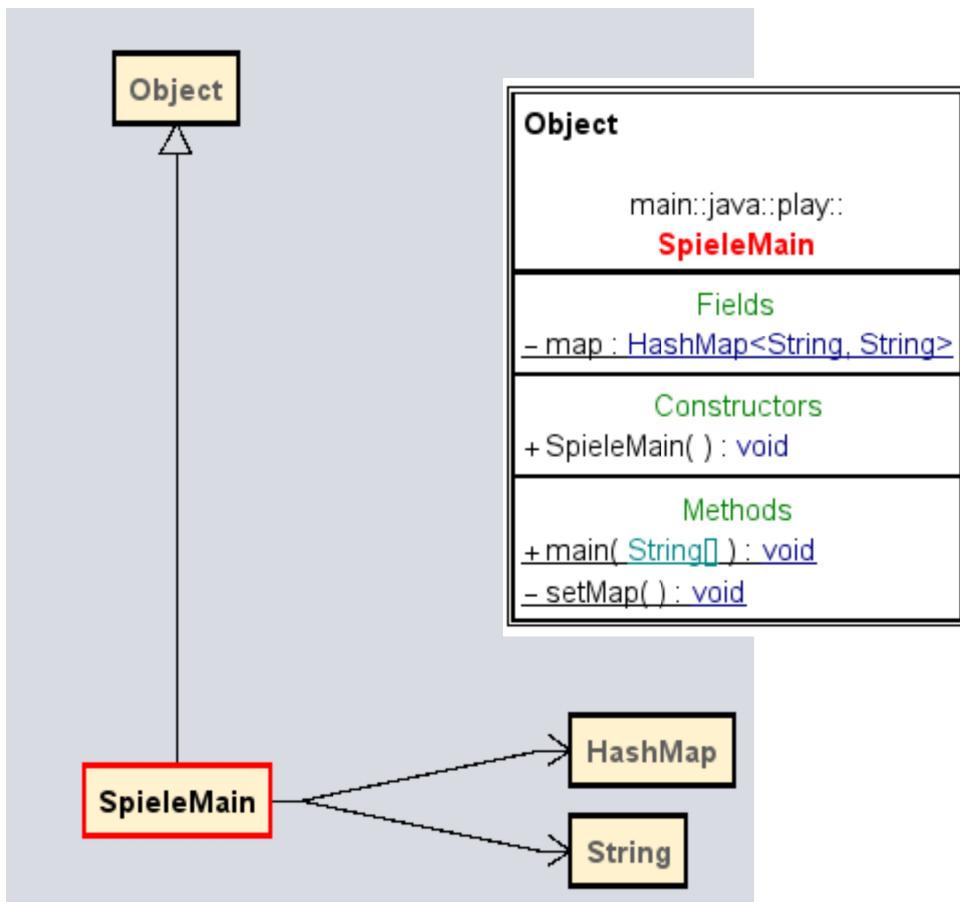
## Aggregates

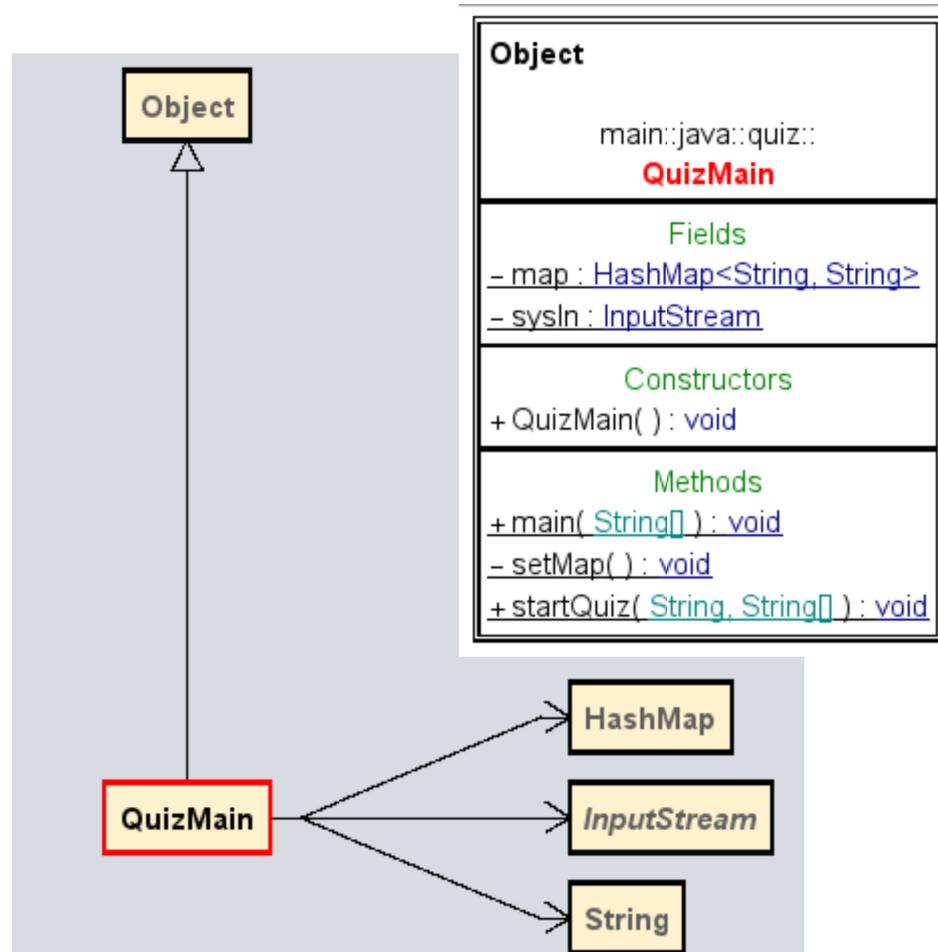
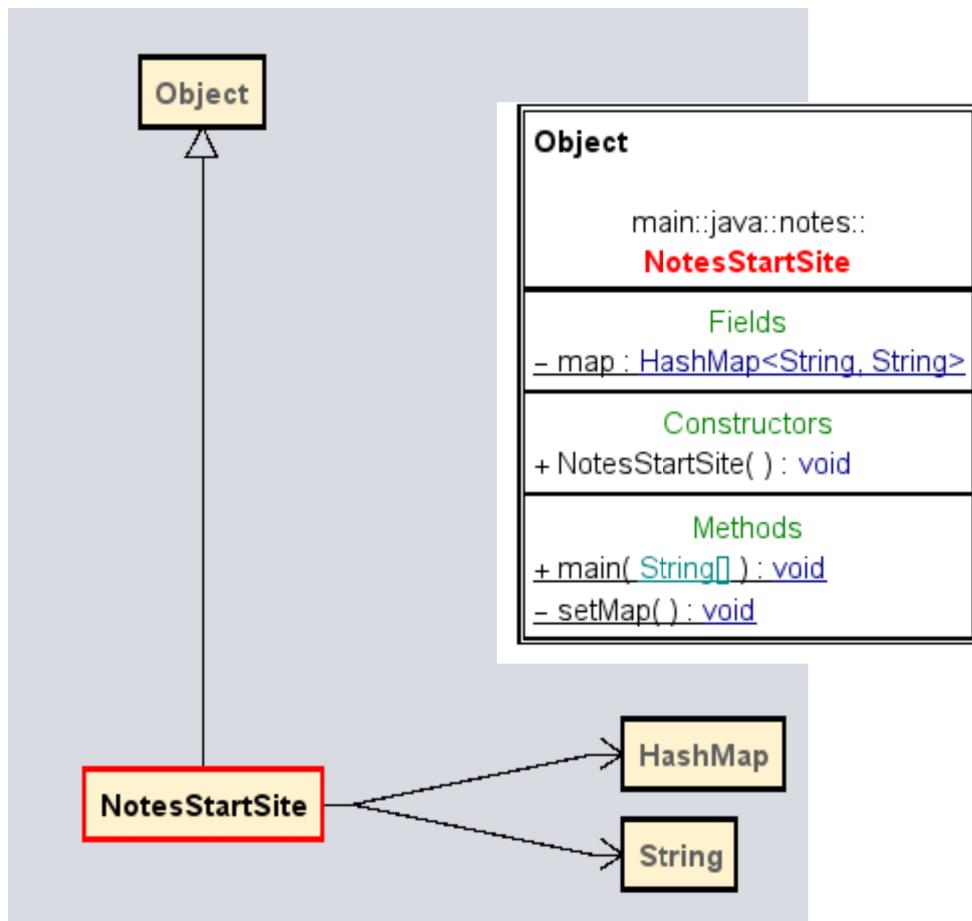
[UML, Beschreibung und Begründung des Einsatzes eines Aggregates; falls kein Aggregate vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

In der vorliegenden Anwendung ist jeder der vier Programmzweige ein eigenes Aggregat. Dies fast die jeweiligen Entitäten zusammen. Jedes Aggregat für sich ist unabhängig von einander ausführbar. Diese können zusätzlich aus dem Hauptmenü, welches ebenfalls ein Aggregat darstellt, gestartet werden. Es wird als Aggregat eingesetzt, da dies die beste Möglichkeit ist die Zweige einfach austauschbar zu machen.

Als UMLs sind jeweils die Startklassen der Zweige angegeben, da ein komplettes Aufzeigen unübersichtlich ist.







# Kapitel 7: Refactoring

## Code Smells

[jeweils 1 Code-Beispiel zu 2 Code Smells aus der Vorlesung; jeweils Code-Beispiel und einen möglichen Lösungsweg bzw. den genommen Lösungsweg beschreiben (inkl. (Pseudo-)Code)]

Code Smell: Large Class

Folgend eine Abbildung der neuen Klasse „Person“. Diese kann den Input des Benutzers abfragen, die Spielpunkte erhöhen, die Punkte ausgeben sowie manuell auf einen bestimmten Wert festlegen. Die Funktionen in dieser als auch in der nächsten Klasse wurden zuvor alle innerhalb der Klasse SchereSteinPapier einzeln genutzt. Es war teilweise unübersichtlich und man konnte nicht sofort sageb, ob der Aufruf zum Computer oder zur spielenden Person gehört. Durch die Erstellung der zwei neuen Klassen kann so durch ein Aufruf des jeweiligen Objekts unterschieden werden, zu wem es letztendlich gehört.

```
public class Person implements User {

    private int points;

    public Person () {
        this.points = 0;
    }

    @Override
    public String getChoice() {

        return HandleInput.getInput(new SchereSteinPapier().getGameMap(), System.in);
    }

    @Override
    public void incrementPoints() {
        this.points++;
    }

    @Override
    public int getUserPoints() {
        return this.points;
    }

    @Override
    public void setUserPoints(int points) {
        this.points = points;
    }
}
```

Folgend eine Abbildung der neuen Klasse „Computer“. Diese hat sie selben Funktionen wie die Klasse Person, nur dass die Methode „getChoice“ keine Eingabe des Nutzers erfordert, sonder eine zufällige Zahl verwendet.

```
public class Computer implements User {

    private int points;

    public Computer() {
        this.points = 0;
    }

    @Override
    public String getChoice() {
        String[] arr = new String[] { "schere", "stein", "papier" };
        int rand = (int) (Math.random() * ((2 - 0) + 1));
        String randChoice = arr[rand];
        return randChoice;
    }

    @Override
    public void incrementPoints() {
        this.points++;
    }

    @Override
    public int getUserPoints() {
        return this.points;
    }

    @Override
    public void setUserPoints(int points) {
        this.points = points;
    }
}
```

Folgend eine Abbildung des neu hinzugefügten Interface, das in den beiden Klassen Person und Computer implementiert wurde.

```
public interface User {

    public String getChoice();

    public void incrementPoints();
    public int getUserPoints();

    public void setUserPoints(int points);
}
```

Auf <https://github.com/0Raspbinary1/Advanced-SE/commit/de424754522b12887>  
74e9dfe3d3aba0262edb560 kann die alte Version eingesehen werden.  
Ein Ausschnitt vor dem Refactoring:

```
int pcPoints = 0;
int userPoints = 0;
int counterRounds = 0;
while (pcPoints < 2 && userPoints < 2) {
    counterRounds++;
    String randChoice = "";
    int rand = 0 + (int) (Math.random() * ((2 - 0) + 1));
    randChoice = arr[rand];
    System.out.println("") + counterRounds + ". Runde: Computer hat sich entschieden.\n"
        + "Was nimmst du? Eingabe in Kleinbuchstaben:\n");

    userInput = HandleInput.getInput(gameMap, System.in);
    System.out.println("userinput: " + userInput + "\n-----");

    System.out.println("user: " + userInput + "\n" + "Computer: " + randChoice + "\n-----");
    int erg = compare(randChoice, userInput);
```

<https://github.com/0Raspbinary1/Advanced-SE/commit/78018360b15e3d83f125f9c32067cb58af893f6>

Ein Ausschnitt nach dem Refactoring:

```
int userPoints = 0;
int counterRounds = 0;
while (pcPoints < 2 && userPoints < 2) {

    while (this.computer.getUserPoints() < 2 && this.person.getUserPoints() < 2) {
        counterRounds++;
        String randChoice = "";
        int rand = 0 + (int) (Math.random() * ((2 - 0) + 1));
        randChoice = arr[rand];
        String pcChoice = this.computer.getChoice();
        System.out.println("") + counterRounds + ". Runde: Computer hat sich entschieden.\n"
            + "Was nimmst du? Eingabe in Kleinbuchstaben:\n");

        userInput = HandleInput.getInput(gameMap, System.in);
        //auslagern
        //userInput = HandleInput.getInput(gameMap, System.in);
        userInput = this.person.getChoice();
        System.out.println("userinput: " + userInput + "\n-----");
```

Code Smell: Long Method

<https://github.com/0Raspbinary1/Advanced-SE/commit/0b1eece01100f5a7c687356575b5a6f3770dab2b>

Der Bereich der Zufallszahl wurde zu Beginn mithilfe einer Methode erstellt. Nach dem Refactoring wurde sie in mehrere aufgeteilt, um eine bessere Übersicht zu haben. Es wurde eine für den unteren Wert und eine für den oberen Wert erstellt.

```
+     public void setRangeInput() {  
+  
+         System.out.println("Gib dein Zahlenraum an:\n");  
-         Scanner sc = new Scanner(System.in);  
+  
+         setRangeDown();  
+         setRangeUp();  
+  
+     }  
+
```

## 2 Refactorings

[2 unterschiedliche Refactorings aus der Vorlesung anwenden, begründen, sowie UML vorher/nachher liefern; jeweils auf die Commits verweisen]

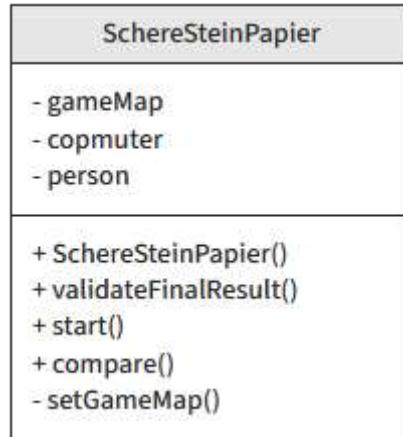
### Refactoring: Extract Method

Vor dem Refactoring gab es nur eine Methode, die das Ergebnis validiert hat und ausgegeben hat, ob man gewonnen hat oder nicht. Jetzt gibt es mehrere Methoden, die jeweils eine eigene Funktion besitzen. Der Grund für die Aufteilung ist die Verkleinerung der ersten Methode zur besseren Übersicht und um die einzelnen Methoden unabhängiger zu machen. Durch das Extrahieren von Code in separate Methoden mit aussagekräftigen Namen kann der Code lesbarer und verständlicher werden. Gut benannte Methoden dienen als Dokumentation für den Code und erleichtern anderen Entwicklern das Verständnis des Codes. Es ist einfacher, den Zweck und die Funktion einer Methode zu verstehen. Des Weiteren hilft es der Reduzierung von Duplikaten. Oftmals wird derselbe Code an mehreren Stellen im Code verwendet. Durch das Extrahieren dieses wiederholten Codes in eine separate Methode kann die Duplikation reduziert werden. Dies führt zu einer verbesserten Wartbarkeit und erleichtert zukünftige Änderungen am Code.

SchereSteinPapier
- gameMap - computer - peron
+ SchereSteinPapier() + compare() + start() - setGameMap()

UML vorher:

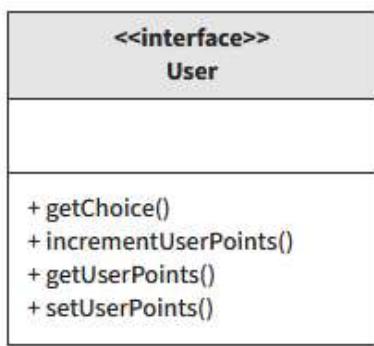
UML nachher



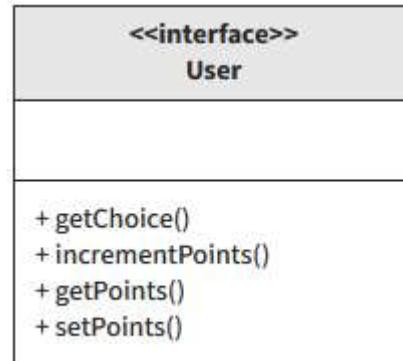
<https://github.com/0Raspbinary1/Advanced-SE/commit/78018360b15e3d83f125f9c32067cb58afd893f6>

### **Refactoring: Rename Method**

Die neue Bezeichnung "incrementPoints" ist präziser und klarer in ihrer Bedeutung. Sie lässt darauf schließen, dass es sich um eine allgemeine Funktion handelt, die Punkte erhöht, ohne auf einen bestimmten Benutzer bezogen zu sein. Die Bezeichnung "incrementUserPoints" könnte unnötig lang oder komplex sein, insbesondere wenn die Funktion nur eine einfache Aufgabe erfüllt, wie z.B. die Erhöhung von Punkten. Durch die Verwendung der kürzeren Bezeichnung "incrementPoints" wird der Code lesbarer und es wird vermieden, dass unnötige Details in der Methodebezeichnung enthalten sind. Durch die Verwendung der allgemeineren Bezeichnung "incrementPoints" kann die Methode potenziell in anderen Kontexten wiederverwendet werden, in denen eine Punkterhöhung erforderlich ist, unabhängig von einem bestimmten Benutzer. Dies verbessert die Flexibilität und Wiederverwendbarkeit des Codes.



UML vorher:



UML nachher

<https://github.com/0Raspbinary1/Advanced-SE/commit/3cf46c4fc60537a5f0a7bedd0f01e73733256111>

# Kapitel 8: Entwurfsmuster

[2 unterschiedliche Entwurfsmuster aus der Vorlesung (oder nach Absprache auch andere) jeweils sinnvoll einsetzen, begründen und UML-Diagramm]

## Entwurfsmuster: Strategie

Die Klasse Conversion im Package main.java.convert wurde mit dem Entwurfsmuster „Strategie“ implementiert. Zu den Gründen zählen Aspekte wie Trennung von Verantwortlichkeiten, Erweiterbarkeit, Wiederverwendbarkeit und Testbarkeit.

Durch die Verwendung des "Strategie"-Musters wird die Verantwortlichkeit für die Durchführung von Konvertierungen von der Klasse "Conversion" auf separate Klassen wie "Weigthing", "Length", "Temperature", "Area" und "Time" übertragen. Jede dieser Klassen ist spezialisiert auf die Konvertierung von Maßeinheiten eines bestimmten Typs (Gewicht, Länge, Temperatur, Fläche, Zeit) und enthält die entsprechende Logik für die Konvertierung. Dadurch wird der Code in der "Conversion"-Klasse entlastet und die Klassen bleiben besser auf ihre spezifischen Aufgaben fokussiert. Ebenso wird die Erweiterbarkeit der Klasse "Conversion" verbessert. Wenn in der Zukunft neue Konvertierungstypen hinzugefügt werden sollen, kann einfach eine neue Klasse erstellt werden, die das entsprechende Interface implementiert und die Konvertierungslogik für den neuen Typ enthält. Die "Conversion"-Klasse muss nicht geändert werden, da sie bereits auf das Interface zugreift. Dadurch wird der Code offen für Erweiterungen und schützt gegen Modifikationen, gemäß dem "Open/Closed"-Prinzip aus dem SOLID-Prinzipien. Des Weiteren wird die Konvertierungslogik in separaten Klassen gekapselt, die unabhängig von der "Conversion"-Klasse wiederverwendet werden können. Wenn in anderen Teilen des Codes Konvertierungen von Maßeinheiten benötigt werden, können die entsprechenden Klassen einfach wiederverwendet werden, ohne dass die Logik zur Konvertierung von Maßeinheiten dupliziert werden muss. Dadurch wird der Code insgesamt konsistenter und einfacher zu warten. Außerdem wird die Testbarkeit der "Conversion"-Klasse verbessert. Da die Konvertierungslogik in separaten Klassen gekapselt ist, können Unit-Tests für jede Klasse einzeln erstellt werden, um sicherzustellen, dass die Konvertierungen korrekt funktionieren. Dadurch wird das Testen einfacher und effizienter, und potenzielle Fehler können frühzeitig erkannt und behoben werden.

Conversion
<ul style="list-style-type: none"> <li>- sc</li> <li>- map</li> <li>- defaultUnits</li> <li>- unit</li> <li>- inputNumber</li> <li>- goalUnit</li> <li>- type</li> <li>- systemIn</li> <li>- output</li> </ul> <ul style="list-style-type: none"> <li>+ Conversion()</li> <li>- setupScanner()</li> <li>+ perform()</li> <li>- convertTemperature()</li> <li>- convert()</li> <li>- askForInputs()</li> <li>+ getOutput()</li> <li>+ setUnit()</li> <li>+ setInputNumber()</li> <li>+ setGoalUnit()</li> <li>+ setSystemIn()</li> <li>- parseMap()</li> <li>- readDoubleInput()</li> <li>- setMap</li> </ul>

## Entwurfsmuster: Builder

In der Klasse HandleNote im Package main.java.notes wurde das Entwurfsmuster „Builder“ verwendet. Folgende Gründe sprechen dafür, einen Builder zu verwenden. Ein Grund, warum der Builder in diesem Code verwendet wurde, ist die Notwendigkeit, ein Objekt mit vielen konfigurierbaren Optionen zu erstellen. Mit dem Builder-Muster können verschiedene Optionen und Eigenschaften eines Objekts durch separate Methoden im Builder selbst konfiguriert werden, anstatt viele Argumente in einem Konstruktor zu übergeben oder komplexe Konfigurationsparameter zu verwenden. Dies führt zu einem saubereren und übersichtlichen Code. Ein weiterer Grund für die Verwendung des Builder-Musters ist die verbesserte Lesbarkeit und Wartbarkeit des Codes. Durch die Verwendung von "sprechenden" Methoden im Builder, die die Konfigurationsschritte repräsentieren, wird der Code leichter verständlich und nachvollziehbar. Es ist auch einfacher, den Code zu erweitern, indem neue Methoden im Builder hinzugefügt werden, um zusätzliche Konfigurationsoptionen zu unterstützen, ohne die zugrunde liegende Klasse zu ändern. Darüber hinaus kann das Builder-Muster die Erstellung von Objekten mit gültigen Zuständen gewährleisten. Der Builder kann Validierungen und Überprüfungen durchführen, um sicherzustellen, dass nur gültige Konfigurationen verwendet werden, bevor das endgültige Objekt erstellt wird. Dies hilft, Fehler und Inkonsistenzen in der Verwendung von Objekten zur Laufzeit zu minimieren. Ein weiterer Vorteil der Verwendung des Builder-Musters ist die Flexibilität bei der Erstellung von Objekten. Mit dem Builder können verschiedene Varianten desselben Objekts erstellt werden, indem unterschiedliche Konfigurationsoptionen verwendet werden. Dies erleichtert die

Erstellung von Objekten mit unterschiedlichen Eigenschaften oder Verhalten, ohne die Klasse selbst zu ändern oder zu überladen.

HandleNote
- fileName - fileMap - numberMap - optionMap - text - isTest - noteList - dir
+ HandleNote() + setTest() + setNoteList() + setFileMap() + perform() + delete() + setFileName() - createNote() + createFile() - getTitle() + writeInFile() + getNoteList() - getNoteText() - show() - list() + getListAsString() - inititateOptionMap() - chooseNote() - performOptionOnNote()

## rework HandleInput

[Browse files](#)

 [main \(#4\)](#)

 [Jennif6r committed on Jan 11](#)

1 parent [668f30c](#) commit [772f548](#)

Showing 3 changed files with 12 additions and 24 deletions.

[Split](#)

[Unified](#)

14 Code/Fun&Learn/src/convert/ConversionStartSite.java

```

...
@@ -1,32 +1,24 @@
1 package convert;
2
3 import java.util.HashMap;
4 -import java.util.Map;
5 -import java.util.Scanner;
6
7 import alles.HandleExit;
8
9 public class ConversionStartSite {
10 -    private static Map<String,
11 String> map = new HashMap<>();
12     public static void
13     main(String[] args) {
14         setMap();
15         Scanner sc = new
16             Scanner(System.in);
17         String input;
18         String text = "Was
19 möchten Sie umrechnen? \n 1: Gewicht \n
2: Längenangaben \n 3: Temperaturen \n
4: Fläche \n"
20             + " 5:
21 Zeit \n S: Zum Start zurück \n X:
22 Programm beenden \n...";
23
24         System.out.println(text);
25         input =
26             sc.nextLine().toUpperCase();
27         while(!map.containsKey(input)) {
28             System.out.println("Falsche Eingabe,
29 versuch es erneut:");
30             input =
31                 sc.nextLine().toUpperCase();
32         }
33
34         if(map.get(input).equals("Start") ||
35 map.get(input).equals("Exit")) {
36             HandleExit.perform(map.get(input),
37                 args);
38         } else {
39             new
40                 Conversion(map.get(input), args);
41         }
42     }
43     private static void
44     setMap() {
45         map.put("1", "Gewicht");
46         map.put("2", "Längenangaben");
47         map.put("3", "Temperaturen");
48         map.put("4", "Fläche");
49         map.put("5", "Zeit");
50         map.put("S", "Zum Start zurück");
51         map.put("X", "Programm beenden");
52     }
53 }

```

```

1 package convert;
2
3 import java.util.HashMap;
4
5 import alles.HandleExit;
6 +import alles.HandleInput;
7
8 public class ConversionStartSite {
9 +    private static HashMap<String,
10 String> map = new HashMap<>();
11     public static void
12     main(String[] args) {
13         setMap();
14
15         String text = "Was
16 möchten Sie umrechnen? \n 1: Gewicht \n
17 2: Längenangaben \n 3: Temperaturen \n
18 4: Fläche \n"
19             + " 5:
20 Zeit \n S: Zum Start zurück \n X:
21 Programm beenden \n...";
22
23         System.out.println(text);
24         String input =
25             HandleInput.getInput(map);
26
27         if(map.get(input).equals("Start") ||
28 map.get(input).equals("Exit")) {
29             HandleExit.perform(map.get(input),
30                 args);
31         } else {
32             new
33                 Conversion(map.get(input), args);
34         }
35     }
36     private static void
37     setMap() {
38         map.put("1", "Gewicht");
39         map.put("2", "Längenangaben");
40         map.put("3", "Temperaturen");
41         map.put("4", "Fläche");
42         map.put("5", "Zeit");
43         map.put("S", "Zum Start zurück");
44         map.put("X", "Programm beenden");
45     }
46 }

```

```
28
29     -         sc.close();
30     }
31
32     private static void setMap() {
```

```
21
22     }
23
24     private static void setMap() {
```

▽ 8 Code/Fun&Learn/src/notes/HandleNote.java □

```
126
127     private void
128     initiateOptionMap() {
129
130         this.optionMap.put("A", "show");
131
132         this.optionMap.put("L", "delete");
133
134         this.optionMap.put("N", "new");
135
136         this.optionMap.put("X", "Exit");
137
138         this.optionMap.put("S", "Start");
139     }
140
141     case "new":
142
143         createNote();
144
145         break;
146
147     }
148
149 }
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
```

```
126
127     private void
128     initiateOptionMap() {
129
130         this.optionMap.put("A", "show");
131
132         this.optionMap.put("D", "delete");
133
134         this.optionMap.put("N", "new");
135
136         this.optionMap.put("L", "list");
137
138         this.optionMap.put("X", "Exit");
139
140         this.optionMap.put("S", "Start");
141     }
142
143     case "new":
144
145         createNote();
146
147         break;
148
149     case "list":
150
151         list(args);
152
153         break;
154
155     }
```

▽ 14 Code/Fun&Learn/src/notes/NotesStartSite.java □

```
...
1    @@ -1,33 +1,25 @@
2
3     package notes;
4
5     import java.util.HashMap;
6
7     import java.util.Map;
8
9     import java.util.Scanner;
10
11    import alles.HandleExit;
```

```
1
2
3     package notes;
4
5     import java.util.HashMap;
6
7     import alles.HandleExit;
8
9     public class NotesStartSite {
10
11     private static Map<String,
```

```
1
2
3     package notes;
4
5     import java.util.HashMap;
6
7     import alles.HandleInput;
8
9     public class NotesStartSite {
10
11     private static HashMap<String,
```

```

12     public static void
13         main(String[] args) {
14             setMap();
15             Scanner sc = new
16                 Scanner(System.in);
17             String input;
18             String text = "Bitte
19                 wähle einen Modus aus:\n 1: Neue Notiz
20                 erstellen\n 2: Notizen auflisten\n S:
21                 Zum Start zurück\n "
22             + "X:
23                 Programm beenden \n...";
24
25             System.out.println(text);
26
27             input =
28                 sc.nextLine().toUpperCase();
29
30             while(!map.containsKey(input)) {
31
32                 System.out.println("Falsche Eingabe,
33                     versuch es erneut:");
34
35                 input =
36                     sc.nextLine().toUpperCase();
37
38             }
39
40             if(map.get(input).equals("Start") ||
41                 map.get(input).equals("Exit")) {
42
43                 HandleExit.perform(map.get(input),
44                     args);
45
46             } else {
47
48                 new
49                     HandleNote(map.get(input), args);
50
51             }
52
53             sc.close();
54
55         }
56
57         private static void setMap() {
58             map.put("1", "new");
59
60         }
61
62     }
63
64     public static void
65         main(String[] args) {
66             setMap();
67
68             String text = "Bitte
69                 wähle einen Modus aus:\n 1: Neue Notiz
70                 erstellen\n 2: Notizen auflisten\n S:
71                 Zum Start zurück\n "
72             + "X:
73                 Programm beenden \n...";
74
75             System.out.println(text);
76
77             String input =
78                 HandleInput.getInput(map);
79
80             if(map.get(input).equals("Start") ||
81                 map.get(input).equals("Exit")) {
82
83                 HandleExit.perform(map.get(input),
84                     args);
85
86             } else {
87
88                 new
89                     HandleNote(map.get(input), args);
90
91             }
92
93         }
94
95         private static void setMap() {
96             map.put("1", "new");
97
98         }
99
100    }
101
102    public static void
103        main(String[] args) {
104            setMap();
105
106            Scanner sc = new
107                Scanner(System.in);
108            String input;
109            String text = "Bitte
110                wähle einen Modus aus:\n 1: Neue Notiz
111                erstellen\n 2: Notizen auflisten\n S:
112                Zum Start zurück\n "
113            + "X:
114                Programm beenden \n...";
115
116            System.out.println(text);
117
118            + String input =
119                HandleInput.getInput(map);
120
121            if(map.get(input).equals("Start") ||
122                map.get(input).equals("Exit")) {
123
124                HandleExit.perform(map.get(input),
125                    args);
126
127            } else {
128
129                new
130                    HandleNote(map.get(input), args);
131
132            }
133
134        }
135
136        private static void setMap() {
137            map.put("1", "new");
138
139        }
140
141    }
142
143    public static void
144        main(String[] args) {
145            setMap();
146
147            Scanner sc = new
148                Scanner(System.in);
149            String input;
150            String text = "Bitte
151                wähle einen Modus aus:\n 1: Neue Notiz
152                erstellen\n 2: Notizen auflisten\n S:
153                Zum Start zurück\n "
154            + "X:
155                Programm beenden \n...";
156
157            System.out.println(text);
158
159            + String input =
160                HandleInput.getInput(map);
161
162            if(map.get(input).equals("Start") ||
163                map.get(input).equals("Exit")) {
164
165                HandleExit.perform(map.get(input),
166                    args);
167
168            } else {
169
170                new
171                    HandleNote(map.get(input), args);
172
173            }
174
175        }
176
177        private static void setMap() {
178            map.put("1", "new");
179
180        }
181
182    }
183
184    public static void
185        main(String[] args) {
186            setMap();
187
188            Scanner sc = new
189                Scanner(System.in);
190            String input;
191            String text = "Bitte
192                wähle einen Modus aus:\n 1: Neue Notiz
193                erstellen\n 2: Notizen auflisten\n S:
194                Zum Start zurück\n "
195            + "X:
196                Programm beenden \n...";
197
198            System.out.println(text);
199
200            + String input =
201                HandleInput.getInput(map);
202
203            if(map.get(input).equals("Start") ||
204                map.get(input).equals("Exit")) {
205
206                HandleExit.perform(map.get(input),
207                    args);
208
209            } else {
210
211                new
212                    HandleNote(map.get(input), args);
213
214            }
215
216        }
217
218        private static void setMap() {
219            map.put("1", "new");
220
221        }
222
223    }
224
225    public static void
226        main(String[] args) {
227            setMap();
228
229            Scanner sc = new
230                Scanner(System.in);
231            String input;
232            String text = "Bitte
233                wähle einen Modus aus:\n 1: Neue Notiz
234                erstellen\n 2: Notizen auflisten\n S:
235                Zum Start zurück\n "
236            + "X:
237                Programm beenden \n...";
238
239            System.out.println(text);
240
241            + String input =
242                HandleInput.getInput(map);
243
244            if(map.get(input).equals("Start") ||
245                map.get(input).equals("Exit")) {
246
247                HandleExit.perform(map.get(input),
248                    args);
249
250            } else {
251
252                new
253                    HandleNote(map.get(input), args);
254
255            }
256
257        }
258
259        private static void setMap() {
260            map.put("1", "new");
261
262        }
263
264    }
265
266    public static void
267        main(String[] args) {
268            setMap();
269
270            Scanner sc = new
271                Scanner(System.in);
272            String input;
273            String text = "Bitte
274                wähle einen Modus aus:\n 1: Neue Notiz
275                erstellen\n 2: Notizen auflisten\n S:
276                Zum Start zurück\n "
277            + "X:
278                Programm beenden \n...";
279
280            System.out.println(text);
281
282            + String input =
283                HandleInput.getInput(map);
284
285            if(map.get(input).equals("Start") ||
286                map.get(input).equals("Exit")) {
287
288                HandleExit.perform(map.get(input),
289                    args);
290
291            } else {
292
293                new
294                    HandleNote(map.get(input), args);
295
296            }
297
298        }
299
300        private static void setMap() {
301            map.put("1", "new");
302
303        }
304
305    }
306
307    public static void
308        main(String[] args) {
309            setMap();
310
311            Scanner sc = new
312                Scanner(System.in);
313            String input;
314            String text = "Bitte
315                wähle einen Modus aus:\n 1: Neue Notiz
316                erstellen\n 2: Notizen auflisten\n S:
317                Zum Start zurück\n "
318            + "X:
319                Programm beenden \n...";
320
321            System.out.println(text);
322
323            + String input =
324                HandleInput.getInput(map);
325
326            if(map.get(input).equals("Start") ||
327                map.get(input).equals("Exit")) {
328
329                HandleExit.perform(map.get(input),
330                    args);
331
332            } else {
333
334                new
335                    HandleNote(map.get(input), args);
336
337            }
338
339        }
340
341        private static void setMap() {
342            map.put("1", "new");
343
344        }
345
346    }
347
348    public static void
349        main(String[] args) {
350            setMap();
351
352            Scanner sc = new
353                Scanner(System.in);
354            String input;
355            String text = "Bitte
356                wähle einen Modus aus:\n 1: Neue Notiz
357                erstellen\n 2: Notizen auflisten\n S:
358                Zum Start zurück\n "
359            + "X:
360                Programm beenden \n...";
361
362            System.out.println(text);
363
364            + String input =
365                HandleInput.getInput(map);
366
367            if(map.get(input).equals("Start") ||
368                map.get(input).equals("Exit")) {
369
370                HandleExit.perform(map.get(input),
371                    args);
372
373            } else {
374
375                new
376                    HandleNote(map.get(input), args);
377
378            }
379
380        }
381
382        private static void setMap() {
383            map.put("1", "new");
384
385        }
386
387    }
388
389    public static void
390        main(String[] args) {
391            setMap();
392
393            Scanner sc = new
394                Scanner(System.in);
395            String input;
396            String text = "Bitte
397                wähle einen Modus aus:\n 1: Neue Notiz
398                erstellen\n 2: Notizen auflisten\n S:
399                Zum Start zurück\n "
400            + "X:
401                Programm beenden \n...";
402
403            System.out.println(text);
404
405            + String input =
406                HandleInput.getInput(map);
407
408            if(map.get(input).equals("Start") ||
409                map.get(input).equals("Exit")) {
410
411                HandleExit.perform(map.get(input),
412                    args);
413
414            } else {
415
416                new
417                    HandleNote(map.get(input), args);
418
419            }
420
421        }
422
423        private static void setMap() {
424            map.put("1", "new");
425
426        }
427
428    }
429
430    public static void
431        main(String[] args) {
432            setMap();
433
434            Scanner sc = new
435                Scanner(System.in);
436            String input;
437            String text = "Bitte
438                wähle einen Modus aus:\n 1: Neue Notiz
439                erstellen\n 2: Notizen auflisten\n S:
440                Zum Start zurück\n "
441            + "X:
442                Programm beenden \n...";
443
444            System.out.println(text);
445
446            + String input =
447                HandleInput.getInput(map);
448
449            if(map.get(input).equals("Start") ||
450                map.get(input).equals("Exit")) {
451
452                HandleExit.perform(map.get(input),
453                    args);
454
455            } else {
456
457                new
458                    HandleNote(map.get(input), args);
459
460            }
461
462        }
463
464        private static void setMap() {
465            map.put("1", "new");
466
467        }
468
469    }
470
471    public static void
472        main(String[] args) {
473            setMap();
474
475            Scanner sc = new
476                Scanner(System.in);
477            String input;
478            String text = "Bitte
479                wähle einen Modus aus:\n 1: Neue Notiz
480                erstellen\n 2: Notizen auflisten\n S:
481                Zum Start zurück\n "
482            + "X:
483                Programm beenden \n...";
484
485            System.out.println(text);
486
487            + String input =
488                HandleInput.getInput(map);
489
490            if(map.get(input).equals("Start") ||
491                map.get(input).equals("Exit")) {
492
493                HandleExit.perform(map.get(input),
494                    args);
495
496            } else {
497
498                new
499                    HandleNote(map.get(input), args);
500
501            }
502
503        }
504
505        private static void setMap() {
506            map.put("1", "new");
507
508        }
509
510    }
511
512    public static void
513        main(String[] args) {
514            setMap();
515
516            Scanner sc = new
517                Scanner(System.in);
518            String input;
519            String text = "Bitte
520                wähle einen Modus aus:\n 1: Neue Notiz
521                erstellen\n 2: Notizen auflisten\n S:
522                Zum Start zurück\n "
523            + "X:
524                Programm beenden \n...";
525
526            System.out.println(text);
527
528            + String input =
529                HandleInput.getInput(map);
530
531            if(map.get(input).equals("Start") ||
532                map.get(input).equals("Exit")) {
533
534                HandleExit.perform(map.get(input),
535                    args);
536
537            } else {
538
539                new
540                    HandleNote(map.get(input), args);
541
542            }
543
544        }
545
546        private static void setMap() {
547            map.put("1", "new");
548
549        }
550
551    }
552
553    public static void
554        main(String[] args) {
555            setMap();
556
557            Scanner sc = new
558                Scanner(System.in);
559            String input;
560            String text = "Bitte
561                wähle einen Modus aus:\n 1: Neue Notiz
562                erstellen\n 2: Notizen auflisten\n S:
563                Zum Start zurück\n "
564            + "X:
565                Programm beenden \n...";
566
567            System.out.println(text);
568
569            + String input =
570                HandleInput.getInput(map);
571
572            if(map.get(input).equals("Start") ||
573                map.get(input).equals("Exit")) {
574
575                HandleExit.perform(map.get(input),
576                    args);
577
578            } else {
579
580                new
581                    HandleNote(map.get(input), args);
582
583            }
584
585        }
586
587        private static void setMap() {
588            map.put("1", "new");
589
590        }
591
592    }
593
594    public static void
595        main(String[] args) {
596            setMap();
597
598            Scanner sc = new
599                Scanner(System.in);
600            String input;
601            String text = "Bitte
602                wähle einen Modus aus:\n 1: Neue Notiz
603                erstellen\n 2: Notizen auflisten\n S:
604                Zum Start zurück\n "
605            + "X:
606                Programm beenden \n...";
607
608            System.out.println(text);
609
609            + String input =
610                HandleInput.getInput(map);
611
612            if(map.get(input).equals("Start") ||
613                map.get(input).equals("Exit")) {
614
615                HandleExit.perform(map.get(input),
616                    args);
617
618            } else {
619
620                new
621                    HandleNote(map.get(input), args);
622
623            }
624
625        }
626
627        private static void setMap() {
628            map.put("1", "new");
629
630        }
631
632    }
633
634    public static void
635        main(String[] args) {
636            setMap();
637
638            Scanner sc = new
639                Scanner(System.in);
640            String input;
641            String text = "Bitte
642                wähle einen Modus aus:\n 1: Neue Notiz
643                erstellen\n 2: Notizen auflisten\n S:
644                Zum Start zurück\n "
645            + "X:
646                Programm beenden \n...";
646
647            System.out.println(text);
648
649            + String input =
650                HandleInput.getInput(map);
651
652            if(map.get(input).equals("Start") ||
653                map.get(input).equals("Exit")) {
654
655                HandleExit.perform(map.get(input),
656                    args);
657
658            } else {
659
660                new
661                    HandleNote(map.get(input), args);
662
663            }
664
665        }
666
667        private static void setMap() {
668            map.put("1", "new");
669
670        }
671
672    }
673
674    public static void
675        main(String[] args) {
676            setMap();
677
678            Scanner sc = new
679                Scanner(System.in);
680            String input;
681            String text = "Bitte
682                wähle einen Modus aus:\n 1: Neue Notiz
683                erstellen\n 2: Notizen auflisten\n S:
684                Zum Start zurück\n "
685            + "X:
686                Programm beenden \n...";
686
687            System.out.println(text);
688
689            + String input =
690                HandleInput.getInput(map);
691
692            if(map.get(input).equals("Start") ||
693                map.get(input).equals("Exit")) {
694
695                HandleExit.perform(map.get(input),
696                    args);
697
698            } else {
699
700                new
701                    HandleNote(map.get(input), args);
702
703            }
704
705        }
706
707        private static void setMap() {
708            map.put("1", "new");
709
710        }
711
712    }
713
714    public static void
715        main(String[] args) {
716            setMap();
717
718            Scanner sc = new
719                Scanner(System.in);
720            String input;
721            String text = "Bitte
722                wähle einen Modus aus:\n 1: Neue Notiz
723                erstellen\n 2: Notizen auflisten\n S:
724                Zum Start zurück\n "
725            + "X:
726                Programm beenden \n...";
726
727            System.out.println(text);
728
729            + String input =
730                HandleInput.getInput(map);
731
732            if(map.get(input).equals("Start") ||
733                map.get(input).equals("Exit")) {
734
735                HandleExit.perform(map.get(input),
736                    args);
737
738            } else {
739
740                new
741                    HandleNote(map.get(input), args);
742
743            }
744
745        }
746
747        private static void setMap() {
748            map.put("1", "new");
749
750        }
751
752    }
753
754    public static void
755        main(String[] args) {
756            setMap();
757
758            Scanner sc = new
759                Scanner(System.in);
760            String input;
761            String text = "Bitte
762                wähle einen Modus aus:\n 1: Neue Notiz
763                erstellen\n 2: Notizen auflisten\n S:
764                Zum Start zurück\n "
765            + "X:
766                Programm beenden \n...";
766
767            System.out.println(text);
768
769            + String input =
770                HandleInput.getInput(map);
771
772            if(map.get(input).equals("Start") ||
773                map.get(input).equals("Exit")) {
774
775                HandleExit.perform(map.get(input),
776                    args);
777
778            } else {
779
780                new
781                    HandleNote(map.get(input), args);
782
783            }
784
785        }
786
787        private static void setMap() {
788            map.put("1", "new");
789
790        }
791
792    }
793
794    public static void
795        main(String[] args) {
796            setMap();
797
798            Scanner sc = new
799                Scanner(System.in);
800            String input;
801            String text = "Bitte
802                wähle einen Modus aus:\n 1: Neue Notiz
803                erstellen\n 2: Notizen auflisten\n S:
804                Zum Start zurück\n "
805            + "X:
806                Programm beenden \n...";
806
807            System.out.println(text);
808
809            + String input =
810                HandleInput.getInput(map);
811
812            if(map.get(input).equals("Start") ||
813                map.get(input).equals("Exit")) {
814
815                HandleExit.perform(map.get(input),
816                    args);
817
818            } else {
819
820                new
821                    HandleNote(map.get(input), args);
822
823            }
824
825        }
826
827        private static void setMap() {
828            map.put("1", "new");
829
830        }
831
832    }
833
834    public static void
835        main(String[] args) {
836            setMap();
837
838            Scanner sc = new
839                Scanner(System.in);
840            String input;
841            String text = "Bitte
842                wähle einen Modus aus:\n 1: Neue Notiz
843                erstellen\n 2: Notizen auflisten\n S:
844                Zum Start zurück\n "
845            + "X:
846                Programm beenden \n...";
846
847            System.out.println(text);
848
849            + String input =
850                HandleInput.getInput(map);
851
852            if(map.get(input).equals("Start") ||
853                map.get(input).equals("Exit")) {
854
855                HandleExit.perform(map.get(input),
856                    args);
857
858            } else {
859
860                new
861                    HandleNote(map.get(input), args);
862
863            }
864
865        }
866
867        private static void setMap() {
868            map.put("1", "new");
869
870        }
871
872    }
873
874    public static void
875        main(String[] args) {
876            setMap();
877
878            Scanner sc = new
879                Scanner(System.in);
880            String input;
881            String text = "Bitte
882                wähle einen Modus aus:\n 1: Neue Notiz
883                erstellen\n 2: Notizen auflisten\n S:
884                Zum Start zurück\n "
885            + "X:
886                Programm beenden \n...";
886
887            System.out.println(text);
888
889            + String input =
890                HandleInput.getInput(map);
891
892            if(map.get(input).equals("Start") ||
893                map.get(input).equals("Exit")) {
894
895                HandleExit.perform(map.get(input),
896                    args);
897
898            } else {
899
900                new
901                    HandleNote(map.get(input), args);
902
903            }
904
905        }
906
907        private static void setMap() {
908            map.put("1", "new");
909
910        }
911
912    }
913
914    public static void
915        main(String[] args) {
916            setMap();
917
918            Scanner sc = new
919                Scanner(System.in);
920            String input;
921            String text = "Bitte
922                wähle einen Modus aus:\n 1: Neue Notiz
923                erstellen\n 2: Notizen auflisten\n S:
924                Zum Start zurück\n "
925            + "X:
926                Programm beenden \n...";
926
927            System.out.println(text);
928
929            + String input =
930                HandleInput.getInput(map);
931
932            if(map.get(input).equals("Start") ||
933                map.get(input).equals("Exit")) {
934
935                HandleExit.perform(map.get(input),
936                    args);
937
938            } else {
939
940                new
941                    HandleNote(map.get(input), args);
942
943            }
944
945        }
946
947        private static void setMap() {
948            map.put("1", "new");
949
950        }
951
952    }
953
954    public static void
955        main(String[] args) {
956            setMap();
957
958            Scanner sc = new
959                Scanner(System.in);
960            String input;
961            String text = "Bitte
962                wähle einen Modus aus:\n 1: Neue Notiz
963                erstellen\n 2: Notizen auflisten\n S:
964                Zum Start zurück\n "
965            + "X:
966                Programm beenden \n...";
966
967            System.out.println(text);
968
969            + String input =
970                HandleInput.getInput(map);
971
972            if(map.get(input).equals("Start") ||
973                map.get(input).equals("Exit")) {
974
975                HandleExit.perform(map.get(input),
976                    args);
977
978            } else {
979
980                new
981                    HandleNote(map.get(input), args);
982
983            }
984
985        }
986
987        private static void setMap() {
988            map.put("1", "new");
989
990        }
991
992    }
993
994    public static void
995        main(String[] args) {
996            setMap();
997
998            Scanner sc = new
999                Scanner(System.in);
1000            String input;
1001            String text = "Bitte
1002                wähle einen Modus aus:\n 1: Neue Notiz
1003                erstellen\n 2: Notizen auflisten\n S:
1004                Zum Start zurück\n "
1005            + "X:
1006                Programm beenden \n...";
1006
1007            System.out.println(text);
1008
1009            + String input =
1010                HandleInput.getInput(map);
1011
1012            if(map.get(input).equals("Start") ||
1013                map.get(input).equals("Exit")) {
1014
1015                HandleExit.perform(map.get(input),
1016                    args);
1017
1018            } else {
1019
1020                new
1021                    HandleNote(map.get(input), args);
1022
1023            }
1024
1025        }
1026
1027        private static void setMap() {
1028            map.put("1", "new");
1029
1030        }
1031
1032    }
1033
1034    public static void
1035        main(String[] args) {
1036            setMap();
1037
1038            Scanner sc = new
1039                Scanner(System.in);
1040            String input;
1041            String text = "Bitte
1042                wähle einen Modus aus:\n 1: Neue Notiz
1043                erstellen\n 2: Notizen auflisten\n S:
1044                Zum Start zurück\n "
1045            + "X:
1046                Programm beenden \n...";
1046
1047            System.out.println(text);
1048
1049            + String input =
1050                HandleInput.getInput(map);
1051
1052            if(map.get(input).equals("Start") ||
1053                map.get(input).equals("Exit")) {
1054
1055                HandleExit.perform(map.get(input),
1056                    args);
1057
1058            } else {
1059
1060                new
1061                    HandleNote(map.get(input), args);
1062
1063            }
1064
1065        }
1066
1067        private static void setMap() {
1068            map.put("1", "new");
1069
1070        }
1071
1072    }
1073
1074    public static void
1075        main(String[] args) {
1076            setMap();
1077
1078            Scanner sc = new
1079                Scanner(System.in);
1080            String input;
1081            String text = "Bitte
1082                wähle einen Modus aus:\n 1: Neue Notiz
1083                erstellen\n 2: Notizen auflisten\n S:
1084                Zum Start zurück\n "
1085            + "X:
1086                Programm beenden \n...";
1086
1087            System.out.println(text);
1088
1089            + String input =
1090                HandleInput.getInput(map);
1091
1092            if(map.get(input).equals("Start") ||
1093                map.get(input).equals("Exit")) {
1094
1095                HandleExit.perform(map.get(input),
1096                    args);
1097
1098            } else {
1099
1100                new
1101                    HandleNote(map.get(input), args);
1102
1103            }
1104
1105        }
1106
1107        private static void setMap() {
1108            map.put("1", "new");
1109
1110        }
1111
1112    }
1113
1114    public static void
1115        main(String[] args) {
1116            setMap();
1117
1118            Scanner sc = new
1119                Scanner(System.in);
1120            String input;
1121            String text = "Bitte
1122                wähle einen Modus aus:\n 1: Neue Notiz
1123                erstellen\n 2: Notizen auflisten\n S:
1124                Zum Start zurück\n "
1125            + "X:
1126                Programm beenden \n...";
1126
1127            System.out.println(text);
1128
1129            + String input =
1130                HandleInput.getInput(map);
1131
1132            if(map.get(input).equals("Start") ||
1133                map.get(input).equals("Exit")) {
1134
1135                HandleExit.perform(map.get(input),
1136                    args);
1137
1138            } else {
1139
1140                new
1141                    HandleNote(map.get(input), args);
1142
1143            }
1144
1145        }
1146
1147        private static void setMap() {
1148            map.put("1", "new");
1149
1150        }
1151
1152    }
1153
1154    public static void
1155        main(String[] args) {
1156            setMap();
1157
1158            Scanner sc = new
1159                Scanner(System.in);
1160            String input;
1161            String text = "Bitte
1162                wähle einen Modus aus:\n 1: Neue Notiz
1163                erstellen\n 2: Notizen auflisten\n S:
1164                Zum Start zurück\n "
1165            + "X:
1166                Programm beenden \n...";
1166
1167            System.out.println(text);
1168
1169            + String input =
1170                HandleInput.getInput(map);
1171
1172            if(map.get(input).equals("Start") ||
1173                map.get(input).equals("Exit")) {
1174
1175                HandleExit.perform(map.get(input),
1176                    args);
1177
1178            } else {
1179
1180                new
1181                    HandleNote(map.get(input), args);
1182
1183            }
1184
1185        }
1186
1187        private static void setMap() {
1188            map.put("1", "new");
1189
1190        }
1191
1192    }
1193
1194    public static void
1195        main(String[] args) {
1196            setMap();
1197
1198            Scanner sc = new
1199                Scanner(System.in);
1200            String input;
1201            String text = "Bitte
1202                wähle einen Modus aus:\n 1: Neue Notiz
1203                erstellen\n 2: Notizen auflisten\n S:
1204                Zum Start zurück\n "
1205            + "X:
1206                Programm beenden \n...";
1206
1207            System.out.println(text);
1208
1209            + String input =
1210                HandleInput.getInput(map);
1211
1212            if(map.get(input).equals("Start") ||
1213                map.get(input).equals("Exit")) {
1214
1215                HandleExit.perform(map.get(input),
1216                    args);
1217
1218            } else {
1219
1220                new
1221                    HandleNote(map.get(input), args);
1222
1223            }
1224
1225        }
1226
1227        private static void setMap() {
1228            map.put("1", "new");
1229
1230        }
1231
1232    }
1233
1234    public static void
1235        main(String[] args) {
1236            setMap();
1237
1238            Scanner sc = new
1239                Scanner(System.in);
1240            String input;
1241            String text = "Bitte
1242                wähle einen Modus aus:\n 1: Neue Notiz
1243                erstellen\n 2: Notizen auflisten\n S:
1244                Zum Start zurück\n "
1245            + "X:
1246                Programm beenden \n...";
1246
1247            System.out.println(text);
1248
1249            + String input =
1250                HandleInput.getInput(map);
1251
1252            if(map.get(input).equals("Start") ||
1253                map.get(input).equals("Exit")) {
1254
1255                HandleExit.perform(map.get(input),
1256                    args);
1257
1258            } else {
1259
1260                new
1261                    HandleNote(map.get(input), args);
1262
1263            }
1264
1265        }
1266
1267        private static void setMap() {
1268            map.put("1", "new");
1269
1270        }
1271
1272    }
1273
1274    public static void
1275        main(String[] args) {
1276            setMap();
1277
1278            Scanner sc = new
1279                Scanner(System.in);
1280            String input;
1281            String text = "Bitte
1282                wähle einen Modus aus:\n 1: Neue Notiz
1283                erstellen\n 2: Notizen auflisten\n S:
1284                Zum Start zurück\n "
1285            + "X:
1286                Programm beenden \n...";
1286
1287            System.out.println(text);
1288
1289            + String input =
1290                HandleInput.getInput(map);
1291
1292            if(map.get(input).equals("Start") ||
1293                map.get(input).equals("Exit")) {
1294
1295                HandleExit.perform(map.get(input),
1296                    args);
1297
1298            } else {
1299
1300                new
1301                    HandleNote(map.get(input), args);
1302
1303            }
1304
1305        }
1306
1307        private static void setMap() {
1308            map.put("1", "new");
1309
1310        }
1311
1312    }
1313
1314    public static void
1315        main(String[] args) {
1316            setMap();
1317
1318            Scanner sc = new
1319                Scanner(System.in);
1320            String input;
1321            String text = "Bitte
1322                wähle einen Modus aus:\n 1: Neue Notiz
1323                erstellen\n 2: Notizen auflisten\n S:
1324                Zum Start zurück\n "
1325            + "X:
1326                Programm beenden \n...";
1326
1327            System.out.println(text);
1328
1329            + String input =
1330                HandleInput.getInput(map);
1331
1332            if(map.get(input).equals("Start") ||
1333                map.get(input).equals("Exit")) {
1334
1335                HandleExit.perform(map.get(input),
1336                    args);
1337
1338            } else {
1339
1340                new
1341                    HandleNote(map.get(input), args);
1342
1343            }
1344
1345        }
1346
1347        private static void setMap() {
1348            map.put("1", "new");
1349
1350        }
1351
1352    }
1353
1354    public static void
1355        main(String[] args) {
1356            setMap();
1357
1358            Scanner sc = new
1359                Scanner(System.in);
1360            String input;
1361            String text = "Bitte
1362                wähle einen Modus aus:\n 1: Neue Notiz
1363                erstellen\n 2: Notizen auflisten\n S:
1364                Zum Start zurück\n "
1365            + "X:
1366                Programm beenden \n...";
1366
1367            System.out.println(text);
1368
1369            + String input =
1370                HandleInput.getInput(map);
1371
1372            if(map.get(input).equals("Start") ||
1373                map.get(input).equals("Exit")) {
1374
1375                HandleExit.perform(map.get(input),
1376                    args);
1377
1378            } else {
1379
1380                new
1381                    HandleNote(map.get(input), args);
1382
1383            }
1384
1385        }
1386
1387        private static void setMap() {
1388            map.put("1", "new");
1389
1390        }
1391
1392    }
1393
1394    public static void
1395        main(String[] args) {
1396            setMap();
1397
1398            Scanner sc = new
1399                Scanner(System.in);
1400            String input;
1401            String text = "Bitte
1402                wähle einen Modus aus:\n 1: Neue Notiz
1403                erstellen\n 2: Notizen auflisten\n S:
1
```