

Möbelmarkt-Software

Schriftliche Ausarbeitung

erstellt im Rahmen der Veranstaltung: Programmieren, Design und Implementierung von Algorithmen

Studiengruppe: WI\_WS23\_I

Name Studierender: Tom Wolf, Sidney Schmidt, Yasin Mehmet Ördekci, Justin Liebelt, Frederic Valentin Pilz

Anzahl der Wörter: 3309

(inkl. wörtliche Zitate / Fußnoten)

Anzahl der Wörter: 2635

(exkl. wörtliche Zitate / Fußnoten)

Prüfungskommission: Jürgen Rolf

Abgabedatum: 05.03.2024

**Gender-Hinweis**

Zur besseren Lesbarkeit wird in dieser Hausarbeit das generische Maskulinum verwendet. Die in dieser Arbeit verwendeten Personenbezeichnungen beziehen sich – sofern nicht anders kenntlich gemacht – auf alle Geschlechter.

**Inhaltsverzeichnis**

[Abbildungsverzeichnis II](#_Toc160529759)

[Abkürzungsverzeichnis III](#_Toc160529760)

[1.1 Problembeschreibung 1](#_Toc160529761)

[1.2 Darstellung von Lösungsalternativen 2](#_Toc160529762)

[1.2.1 Abstrakte Ebene: Darstellungstechnik 2](#_Toc160529763)

[1.2.2 Konkrete Ebene: Datentyp für Preise 4](#_Toc160529764)

[1.3 Begründete Auswahl einer Alternative 5](#_Toc160529765)

[1.4 Beschreibung des Programms 7](#_Toc160529766)

[1.5 Beispielausgabe des Programms 12](#_Toc160529767)

[1.6 Fazit/Ausblick 17](#_Toc160529768)

[Anhang IV](#_Toc160529769)

[Literaturverzeichnis XII](#_Toc160529770)

# **Abbildungsverzeichnis**

[Abbildung 1: UML-Klassendiagramm der Möbelstruktur 8](#_Toc160495969)

[Abbildung 2: UML-Klassendiagramm der Lagerstruktur 9](#_Toc160495970)

[Abbildung 3: Preis-Ausgabe der Klasse „Lager“ mit und ohne Berücksichtigung des Rabatts 10](#_Toc160495971)

[Abbildung 4: Methoden zum Lesen und Schreiben eines Objekts aus und in eine Datei in der Klasse „Lagersystem“ 11](#_Toc160495972)

[Abbildung 5: Startfenster des Programms. 12](#_Toc160495973)

[Abbildung 6: Ausgabe einer Suche nach einer Kategorie mit Sondereigenschaft. 12](#_Toc160495974)

[Abbildung 7: Ausgabe einer Kombination an Möbeln für einen eingegebenen Betrag. 13](#_Toc160495975)

[Abbildung 8: Setzen eines Rabattes für eine Kategorie. 13](#_Toc160495976)

[Abbildung 9: Setzen eines Preises für eine Möbelart. 14](#_Toc160495977)

[Abbildung 10: Ausgabe des Gesamtwertes für Kategorien, Bereiche, Gesamt Verkaufspreis und der Bereich mit dem größten Gesamtverkaufspreis 14](#_Toc160495978)

[Abbildung 11: Ausgabe des Lagerbestands. 15](#_Toc160495979)

[Abbildung 12: Änderung des Lagerbestands. 15](#_Toc160495980)

[Abbildung 13: Änderung des Lagerbestands. 16](#_Toc160495981)

[Abbildung 14: Änderung des Lagerbestands. 16](#_Toc160495982)

# **Abkürzungsverzeichnis**

GUI *Graphical User Interface (Grafische Benutzeroberfläche)*

UML *Unified Modeling Language*

FXML *JavaFx* *Extensible Markup Language*

CSS *Cascading Style Sheets*

# **1.1 Problembeschreibung**

Die Verwaltung eines umfangreichen Lagerbestands in einem Möbelmarkt erfordert eine effiziente und gut strukturierte Lösung. Hierbei sollen verschiedene Aspekte durch eine sorgfältig gestaltete Klassen-Hierarchie abgebildet werden.

Ein wichtiger Aspekt ist die Kategorisierung der Möbel im Bestand. Diese sind in verschiedene Kategorien unterteilt und jedem Bereich des Möbelmarktes zugeordnet. Es ist entscheidend, diese Struktur im Programm abzubilden, um eine gezielte Verwaltung der Möbel zu ermöglichen. Infolgedessen muss innerhalb der Klassen die verschiedenen Eigenschaften der Möbel in Form von Attributen innerhalb der Klassen abgebildet werden. Eine Besonderheit hierbei stellen die Sondereigenschaften der Möbel dar, die kategoriespezifisch implementiert werden müssen. Diese Sondereigenschaften müssen im Programm berücksichtigt werden, um Möbelstücke der gleichen Möbelart unterscheiden zu können.

Um den Anforderungen gerecht zu werden, muss das Programm eine Vielzahl von Funktionalitäten unterstützen. Dazu gehören unter anderem die Möglichkeit, den Lagerbestand einer bestimmten Möbelart um einen gegebenen Betrag zu verändern, die Berechnung des Gesamt-Verkaufspreises aller Möbel im Lager sowie die Ermittlung des Gesamtpreises aller Möbel eines bestimmten Bereichs oder einer Kategorie. Des Weiteren ist es wichtig, den Bereich zu identifizieren, in dem der größte Gesamtwert der Möbel vorliegt. Zudem sollen für einen bestimmten Betrag zwischen 100 und 600 Euro Listen von Möbeln mit Stückzahlen zusammengestellt werden, die für diesen Betrag erhältlich sind, wobei der aktuelle Lagerbestand berücksichtigt wird. Weitere Funktionalitäten umfassen die Möglichkeit, den Verkaufspreis eines Möbelstücks anzupassen, eine übersichtliche Darstellung aller Möbel im Bestand mit ihren individuellen Sondereigenschaften sowie die Suche nach Möbeln mit einem bestimmten Wert in ihren Sondereigenschaften. Darüber hinaus sollen Sonderaktionen gestartet werden können, bei der alle Möbel einer bestimmten Kategorie um zehn Prozent reduziert werden. Schließlich ist es wichtig, den aktuellen Lagerbestand wieder in die Datei zu schreiben, aus der der Anfangsbestand ermittelt wird, um das Programm danach ordnungsgemäß zu beenden. Analog dazu muss zu Beginn des Programms der Anfangsbestand aus der Datei geladen werden können. Die erfolgreiche Umsetzung dieses Programms erfordert eine sorgfältige Planung und Implementierung, um eine effiziente und benutzerfreundliche Lagerverwaltung für den Möbelmarkt zu gewährleisten.

# **1.2 Darstellung von Lösungsalternativen**

Die Entwicklung von Softwarelösungen erfordert Entscheidungen auf verschiedenen Ebenen, angefangen bei abstrakten Konzepten wie der Darstellungstechnik bis hin zu konkreten Implementierungsdetails wie der Auswahl von Datentypen.

## **1.2.1 Abstrakte Ebene: Darstellungstechnik**

Ein entscheidender Aspekt dieser Entscheidungsfindung ist die Wahl der Darstellungstechnik, die auf verschiedenen Ebenen, wie der Benutzeroberfläche, betrachtet werden kann. In diesem Kontext werden im Folgenden zunächst die Lösungsalternativen für die abstrakte Ebene der Darstellungstechnik, speziell für die grafische Benutzeroberfläche (GUI) und Konsolenanwendungen, analysiert*.* Im Kontext der grafischen Benutzeroberfläche werden darüber hinaus die Potenziale von JavaFX und Swing als Darstellungstechnologien erörtert.

Die GUI repräsentiert eine Schnittstelle, die durch visuelle Elemente wie Schaltflächen, Menüs und Symbole interaktive Benutzererfahrungen ermöglicht. Der Einsatz von GUIs bietet diverse Vorteile. Erstens ermöglichen sie eine intuitivere und benutzerfreundlichere Interaktion, da Nutzer visuelle Elemente nutzen können. Zweitens eröffnen sie eine breitere Palette von Interaktionsmöglichkeiten, wie beispielsweise Drag-and-Drop, Mausgesten und kontextsensitive Menüs, was die Bedienung für Benutzer erleichtert. Drittens erlauben GUIs die Integration von Grafiken und Icons, was die Anpassbarkeit und ästhetische Qualität der Anwendung verbessert.

Jedoch sind auch Nachteile zu berücksichtigen. GUI-Anwendungen neigen dazu, mehr Systemressourcen zu verbrauchen, insbesondere bei grafikintensiven Elementen oder Animationen. Für Entwickler kann die Erstellung von GUIs komplexer sein und eine steilere Lernkurve haben im Vergleich zur Entwicklung von Konsolenanwendungen.

Konsolenanwendungen bieten eine alternative Lösung, die auf einem textbasierten Interface basiert. Ihre Vorteile liegen in der Ressourceneffizienz, da sie keine aufwendigen Grafikelemente bedürfen, in der einfacheren Automatisierung und Integration in Skripte sowie in der stabilen Performance durch geringere Abhängigkeiten. Was die Automatisierung betrifft, bietet eine Konsolenanwendung typischerweise die Möglichkeit einer offenen Automatisierung (Beispiel: beim Hinzufügen von Skripten) im Gegensatz zu einer geschlossenen Automatisierung bei GUIs.

Jedoch gehen auch hiermit Nachteile einher, wie die begrenzte Benutzerinteraktion, die sich auf Texteingabe und -ausgabe beschränkt. Für Endbenutzer, insbesondere ohne technischen Hintergrund, können Konsolenanwendungen abschreckend wirken und eine höhere Lernkurve haben. Zudem können sie komplexe visuelle Informationen nicht so effektiv darstellen wie GUIs, was zu Einschränkungen in der Funktionalität führen kann.

Im Bereich der grafischen Benutzeroberflächen offenbart sich eine facettenreiche Landschaft von Technologien, die sich in ihrer Komplexität, visuellen Gestaltungsmöglichkeiten und weiteren Aspekten erheblich voneinander unterscheiden können. In diesem Kontext wird deutlich, dass der Begriff „GUI“ keineswegs homogen ist. Vielmehr existieren diverse Technologien, die spezifische Charakteristika und Funktionalitäten aufweisen. Die vorliegende Abhandlung fokussiert sich auf die Kontraste und Kriterien, die zwischen den beiden bedeutenden Java-basierten GUI-Technologien, JavaFX und Swing, manifest werden.

JavaFX zeichnet sich durch eine moderne und flexible Benutzeroberfläche aus, die im Vergleich zu Swing fortschrittlichere Gestaltungsmöglichkeiten bietet. Ein weiterer Vorteil liegt in der Integration von Multimediaelementen, wie Audio und Video, was die Entwicklung von Anwendungen mit reichhaltigen Medienfunktionen ermöglicht. Die Nutzung von FXML, CSS und Scene Builder unterstützt zudem die Erstellung ansprechender Benutzeroberflächen.

Jedoch ist zu beachten, dass JavaFX für Einsteiger als komplex empfunden werden kann, insbesondere bei der Umsetzung fortgeschrittener Funktionen wie Animationen oder komplexen Benutzeroberflächen. Die Einrichtung gestaltet sich im Vergleich zu Swing deutlich aufwendiger. Eine weitere Herausforderung liegt in der langsameren Verbreitung im Vergleich zu Swing, was sich in der Quantität und Qualität der vorhandenen Dokumentation widerspiegelt.

Swing repräsentiert eine ältere Technologie mit stabilen und ausgereiften Merkmalen, die eine umfassende Dokumentation für Beginner aufweist. Diese Stabilität und Reife machen es zu einer verlässlichen Wahl für die Entwicklung von GUI-Anwendungen. Die Lernkurve von Swing wird als einfacher betrachtet, insbesondere für Entwickler mit bereits vorhandener Java-Erfahrung.

Jedoch weist Swing einige Nachteile auf, darunter eine etwas veraltete Optik im Vergleich zu moderneren GUI-Technologien wie JavaFX. Dies könnte in Anwendungen mit hohen ästhetischen Anforderungen als nachteilig betrachtet werden. Des Weiteren bietet Swing nicht die gleiche Unterstützung für moderne Designs und Benutzeroberflächen wie JavaFX, was in Anwendungen, die zeitgemäße Ästhetik erfordern, als Einschränkung betrachtet werden kann. Zudem zeigt sich eine begrenzte Unterstützung für Multimediaelemente im Vergleich zu JavaFX, was die Entwicklung von Anwendungen mit reichhaltigen Medienfunktionen erschweren kann.

## **1.2.2 Konkrete Ebene: Datentyp für Preise**

Auf der konkreten Ebene erfordert die Wahl des Attributs für die Darstellung von Preisen in einem Informationssystem eine gründliche Analyse der Vor- und Nachteile zweier grundlegender Datentypen: Fließkommazahlen (wie double oder float) und ganzzahlige Zahlen (wie Integer oder long).

Die Verwendung ganzzahliger Zahlen zur Darstellung von Preisen in Cent weist eine Reihe von Vorzügen auf. Erstens gewährleisten Ganzzahlen eine präzise Berechnung ohne Rundungsfehler, was zu exakteren Ergebnissen führt. Zweitens ermöglichen sie einfachere und weniger fehleranfällige Vergleiche im Kontrast zu Gleitkommazahlen.

Dennoch sind auch Nachteile zu berücksichtigen. Die Notwendigkeit einer zusätzlichen Umrechnung für die Anzeige oder Verarbeitung in anderen Formaten, insbesondere in Benutzeroberflächen, könnte einen erhöhten Umrechnungsaufwand bedeuten. Des Weiteren besteht bei hohen Werten die potenzielle Gefahr von Überlaufproblemen, sofern der gewählte Datentyp nicht ausreichend dimensioniert ist.

Die Verwendung von Fließkommazahlen zur direkten Darstellung von Eurobeträgen bietet gewisse Vorzüge. Die direkte Anzeige und Nutzung ohne zusätzliche Umrechnungen ermöglicht eine leichtere Implementierung. Des Weiteren können in den Gleitkommazahlen weitaus größere Werte bei gleichem Speicherplatz gespeichert werden als in den primitiven Ganzzahl-Datentypen.

Jedoch sind auch hier Nachteile zu konstatieren. Rundungsfehler können auftreten und in Finanzberechnungen zu inkorrekten Ergebnissen führen. Komplexere Vergleiche zwischen Gleitkommazahlen erfordern eine sorgfältige Beachtung von Toleranzen, da die Genauigkeit begrenzt ist. [[1]](#footnote-1)

# **1.3 Begründete Auswahl einer Alternative**

Die Entscheidung für die Darstellungstechnik zwischen den Lösungsalternativen Konsole oder GUI fiel auf die Entwicklung einer GUI-Anwendung. Mehrere Gründe begünstigten diese Wahl.

Zunächst wurde der Ressourcenverbrauch heutzutage als vernachlässigbar eingestuft. Fortschritte in Hard- und Softwaretechnologien haben dazu geführt, dass der zusätzliche Verbrauch von Systemressourcen durch GUI-Anwendungen in der aktuellen technologischen Landschaft weniger signifikant ist.

Dadurch, dass sich GUIs in den letzten Jahrzehnten immer mehr etabliert haben, sind die entsprechenden Frameworks wesentlich ausgereifter und haben auch fast keine Stabilitätsprobleme mehr. Demzufolge ist die Stabilität heutzutage ebenfalls als vernachlässigbar einzustufen.

Zudem bestanden keine Anforderungen oder Pläne für eine offene Automatisierung der Anwendung, weswegen dies bei der Entscheidung gegen eine GUI keine maßgebliche Rolle spielte.

Zuletzt wurde das Nutzererlebnis für Endbenutzer als entscheidender bewertet als der mögliche Mehraufwand für die Entwicklung einer GUI. Insbesondere für Endbenutzer ohne technischen Hintergrund kann die Effizienz durch Interaktion mit visuellen Elementen wie Schaltflächen und Menüs höher ausfallen. Dies war der wichtigste Faktor, der zur Entscheidung einer grafischen Benutzeroberfläche führte.

In Bezug auf die Wahl zwischen JavaFX und Swing fiel die Entscheidung zugunsten von Swing aus. Obwohl JavaFX eine modernere Benutzeroberfläche bietet und die Aufsetzung nicht als übermäßig problematisch angesehen wurde, wurden mehrere Gründe für Swing als ausschlaggebend erachtet.

Erstens wurde die größere Dokumentation und die als einfacher wahrgenommene Umsetzbarkeit von Swing im Vergleich zu JavaFX betont. Insbesondere aus zeitlicher Perspektive wurde die Entscheidung für Swing getroffen, um die Effizienz in der Entwicklungsphase zu maximieren. Die breitere Verfügbarkeit von Ressourcen und das etablierte Wissen in der Entwicklergemeinschaft spielten dabei eine entscheidende Rolle.

In Bezug auf die Auswahl zwischen der Repräsentation von Preisen als Fließkommazahlen oder ganzzahligen Zahlen in Cent wurde zunächst die Entscheidung für ganzzahlige Zahlen getroffen. Diese Entscheidung basiert auf einer sorgfältigen Abwägung verschiedener Faktoren, die im Kontext der gegebenen Problemstellung relevant sind.

Ein maßgeblicher Grund für die Präferenz von ganzzahligen Zahlen liegt in der Einschätzung, dass ein hoher Speicherbedarf und potenzielle Überlaufprobleme in diesem konkreten Anwendungsszenario nicht zu erwarten sind. Diese Einschätzung beruht auf der Tatsache, dass selbst bei der Berücksichtigung von Szenarien mit einem sehr großen Möbelmarkt und Millionen von Möbeln im Lager, die Realisierung dieser extremen Größenordnungen als unrealistisch betrachtet wird. Der Möbelmarkt wird als begrenzt angesehen, und die Stückzahlen würden nicht in einem Ausmaß ansteigen, das Überlaufprobleme verursachen könnte. Die Anwendung von ganzzahligen Zahlen erscheint daher als ökonomischere Wahl, die dennoch ausreichend Genauigkeit bietet.

Ein weiterer bedeutender Grund, der zur Präferenz für ganzzahlige Zahlen führte, ist die höchste Gewichtung der Genauigkeit. Die finanziellen Auswirkungen von Rundungsfehlern könnten zu monetären Nachteilen führen, weshalb die exakte Darstellung von Preisen als essenziell erachtet wird. Unter diesem Gesichtspunkt wird der zusätzliche Entwicklungsaufwand, der mit der Verwendung von ganzzahligen Zahlen einhergeht, als vernachlässigbar betrachtet.

# **1.4 Beschreibung des Programms**

Im vorliegenden Programm zur Verwaltung von Möbelstücken wird eine klare und strukturierte Klassenhierarchie verwendet, um die verschiedenen Aspekte der Möbelrepräsentation und -verwaltung zu modellieren. Die zentrale Mutterklasse ist dabei die Klasse „Moebelstueck“, welche die grundlegenden Attribute für jedes Möbelstück beinhaltet. Anhang 1 zeigt den Anfang dieser Klasse mit den entsprechenden Enums und Attributen sowie dem Konstruktor. Die umfassten Attribute sind der Verkaufspreis, die Kategorie - repräsentiert durch ein Enum mit den vier Möglichkeiten Lagerung, Liegemoebel, Sitzmoebel und Tische - und dem Bereich ebenfalls durch ein Enum mit den Kategorien Küche, Wohnen, Schlafen und Andere. Die Mutterklasse implementiert standardmäßige Getter- und Setter-Methoden, um auf die Attribute zuzugreifen. Zudem wird die Schnittstelle „Serializable“ aus Gründen der Speicherung in einer Datei implementiert. Die Erläuterung dessen folgt am Schlussteil von „1.4 Beschreibung des Programms“.

Die spezialisierten Klassen der Klasse Moebelstueck, nämlich die Tochterklassen, repräsentieren die vier Kategorien: Lagerung, Liegemoebel, Sitzmoebel und Tische. In Anhang 2 wird exemplarisch am Anfang der Klasse „Lagerung“ gezeigt, wie die (eine Kategorie repräsentierenden) Klassen aufgebaut sind. Jede dieser Klassen enthält zusätzliche Attribute wie die Sondereigenschaft (durch ein Enum) und die Möbelart (ebenfalls durch ein Enum). Die Tochterklassen implementieren ebenfalls die notwendigen Getter- und Setter-Methoden sowie eine toString-Methode.

Die gesamte Möbelstruktur, repräsentiert durch ein Unified Modeling Language (UML)-Klassendiagramm, ist in Abbildung 1 ersichtlich.

Ein Bild, das Text, Screenshot, Diagramm, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 1: UML-Klassendiagramm der Möbelstruktur.

Eigene Erstellung mittels IntelliJ Diagrams plugin.

Für jede Möbelart existiert ein einzigartiges Lager (Anhang 3). Diese Klassen erben vom Lager der zu ihrer Möbelart gehörenden Kategorie-Klasse.

Die Lagerklassen der vier existierenden Kategorien (exemplarisch an der Lagerunglager-Klasse in Anhang 4) bieten Methoden zum Hinzufügen und Entfernen von Kategorie-Objekten sowie zur Erstellung von Listen bestimmter Sondereigenschaften und zum Erhalten des Lagerbestands. Des Weiteren bietet Sie Methoden zum Lesen und Schreiben in und aus einer Datei (Anhang 5). Die Mutterklasse aller Kategorie-Lager ist die Klasse „Lager<K>“, welche die generische Identifizierung der Kategorie ermöglicht.

Die zentrale Mutterklasse aller Lager ist die generische und abstrakte Klasse „Lager“ (Anhang 6). Sie beinhaltet Attribute wie den Preis, einen Boolean-Wert, der angibt, ob ein zehn-prozentiger Rabatt gewährt wird, und eine ArrayList von Objekten der jeweiligen Kategorie. Außerdem wird auch hier die Schnittstelle „Serializable“ implementiert.

Die gesamte Lagerstruktur, repräsentiert durch ein UML-Klassendiagramm, ist in Abbildung 2 ersichtlich.

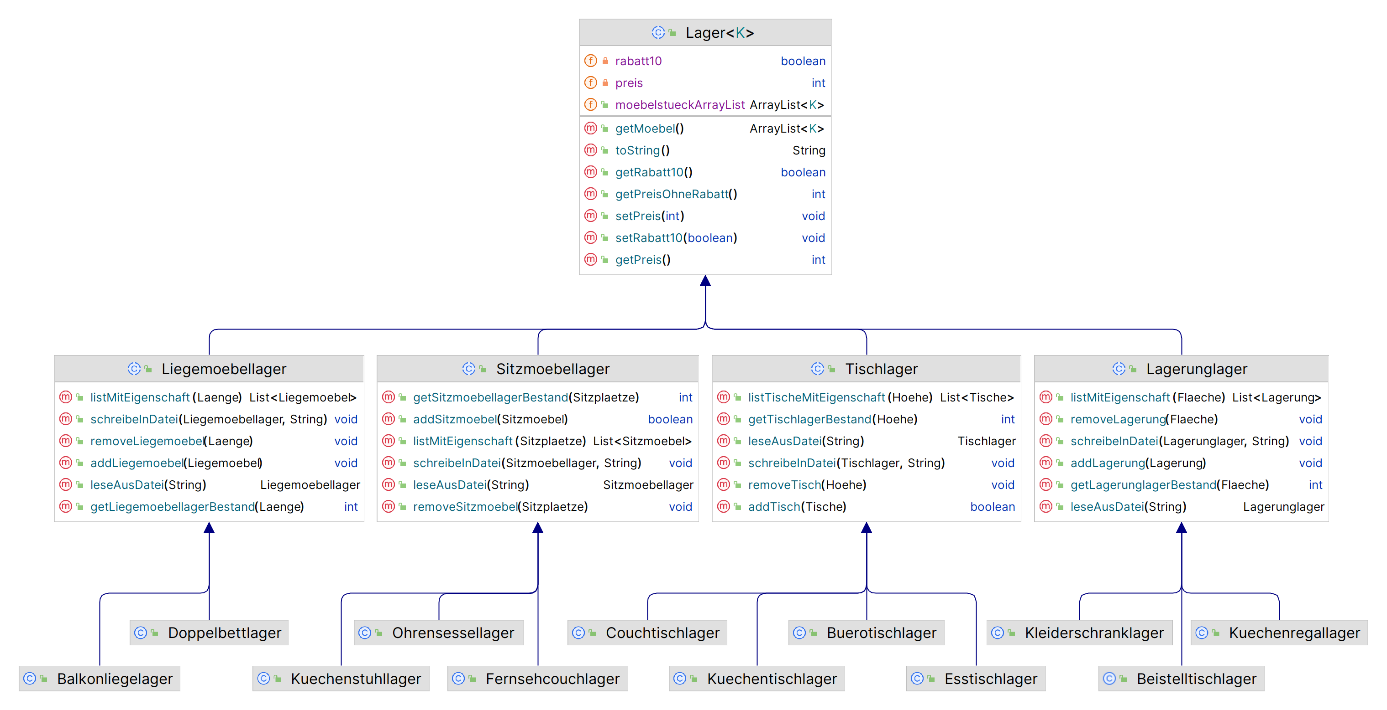


Abbildung 2: UML-Klassendiagramm der Lagerstruktur.

Eigene Erstellung mittels IntelliJ Diagrams plugin.

Der Preis wird dabei mit oder ohne Rabatt ausgegeben (Abbildung 3). Bei der Ausgabe mit Rabatt wird zunächst überprüft, ob „rabatt10“ wahr ist, also eine Rabattaktion mit zehn-prozentigem Rabatt läuft. Wenn nicht, wird der Preis zurückgegeben. Sollte dies aber der Fall sein wird zusätzlich auf das Aufrunden des Cents bei über 0,5 Cent geachtet. Dabei wird durch „preis \* 9 % 10“ die Nachkommastelle berechnet und überprüft, ob sie größer oder gleich fünf ist. Sollte dies der Fall sein wird der Centbetrag, der durch „preis \* 9 / 10“ berechnet wird, um 1 erhöht, ansonsten ohne weitere Bearbeitung zurückgegeben.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 3: Preis-Ausgabe der Klasse „Lager“ mit und ohne Berücksichtigung des Rabatts.

Eigene Erstellung.

Zusätzlich existieren zwei weitere bedeutende Klassen im Programm: „Lagersystem“ (Anhang 7) und „Moebelhaus“ (Anhang 8). Die Klasse „Lagersystem“ enthält die gesamte Logik und die geforderten Funktionalitäten für das Anwendersystem. Sie beinhaltet alle Lager der einzelnen Möbelarten und implementiert Methoden zum Schreiben und Lesen von Dateien (Abbildung 4), sowie Getter- und Setter-Methoden für Dateien und Lagerbestand-Änderungen.

Die Klasse "Moebelhaus" dient als Einstiegspunkt für die grafische Benutzeroberfläche. Sie hält zudem das Lagersystem und lädt die Speicher-Datei.

Im Rahmen des Projektes ist es notwendig den aktuellen Stand des Möbelhauses zu laden und zu speichern. Das setzten wir um, indem wir das gesamte Lagersystem-Objekt und dadurch verknüpfte Klassen abspeichern und laden. Zum Schreiben der Datei werden die Klassen durch die Implementation der Schnittstelle „Serializable“ serialisiert. Über einen „FileOutputStream“ werden sie mittels „ObjectOutputStream“ in einer Datei geschrieben. Zum Lesen der Objekte aus der Datei werden über einen „FileInputStream“ mittels „ObjectInputStream“ die Objekte eingelesen und deserialisiert. Die entsprechenden Methoden sieht man in Abbildung 4 und Anhang 5. [[2]](#footnote-2)

Ein Bild, das Text, Screenshot, Schrift, Dokument enthält.

Automatisch generierte Beschreibung

Abbildung 4: Methoden zum Lesen und Schreiben eines Objekts aus und in eine Datei in der Klasse „Lagersystem“.

Eigene Erstellung.

# **1.5 Beispielausgabe des Programms**

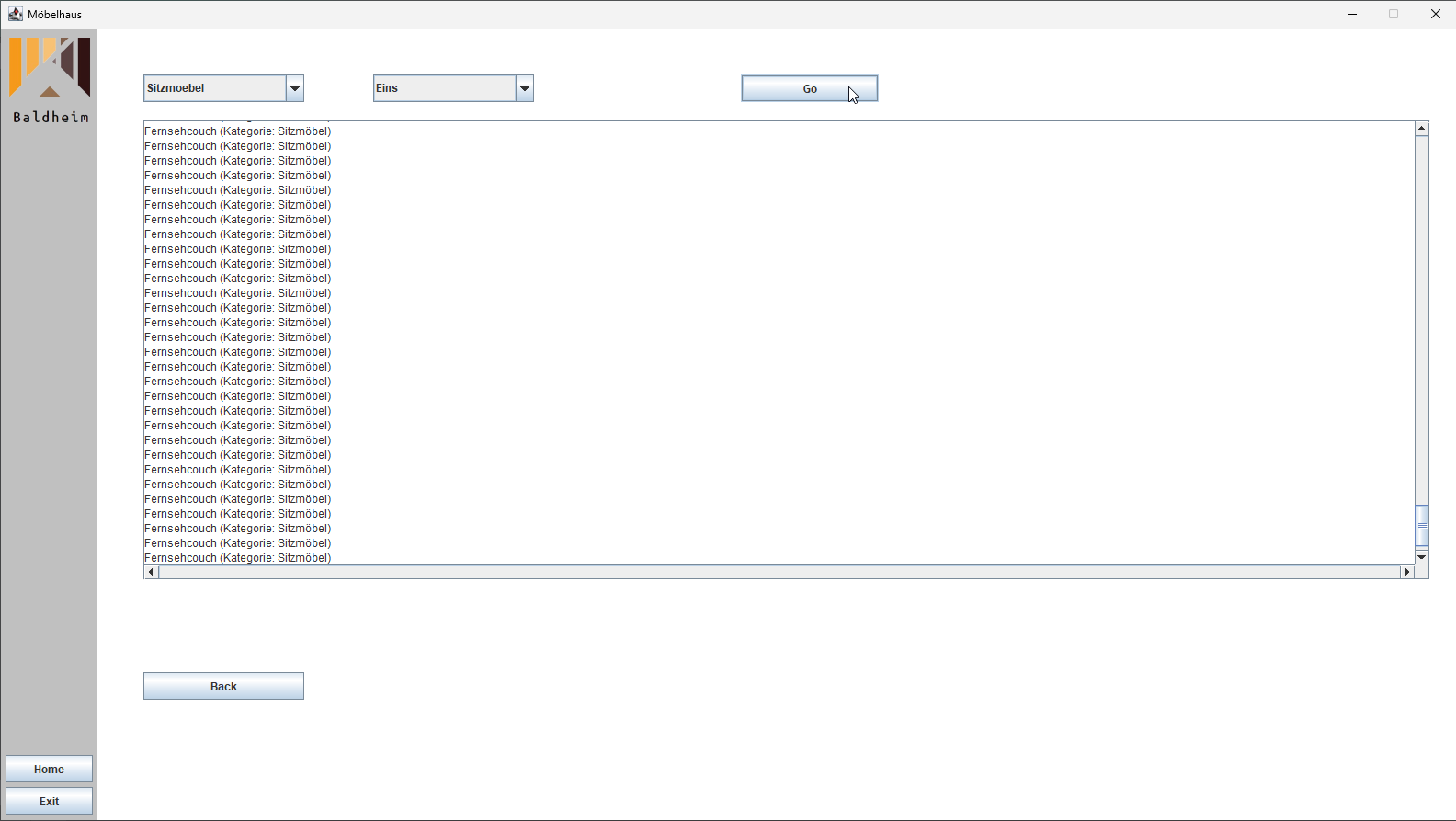


Abbildung 5: Startfenster des Programms.

Eigene Erstellung.

Abbildung 6: Ausgabe einer Suche nach einer Kategorie mit Sondereigenschaft.

Eigene Erstellung.

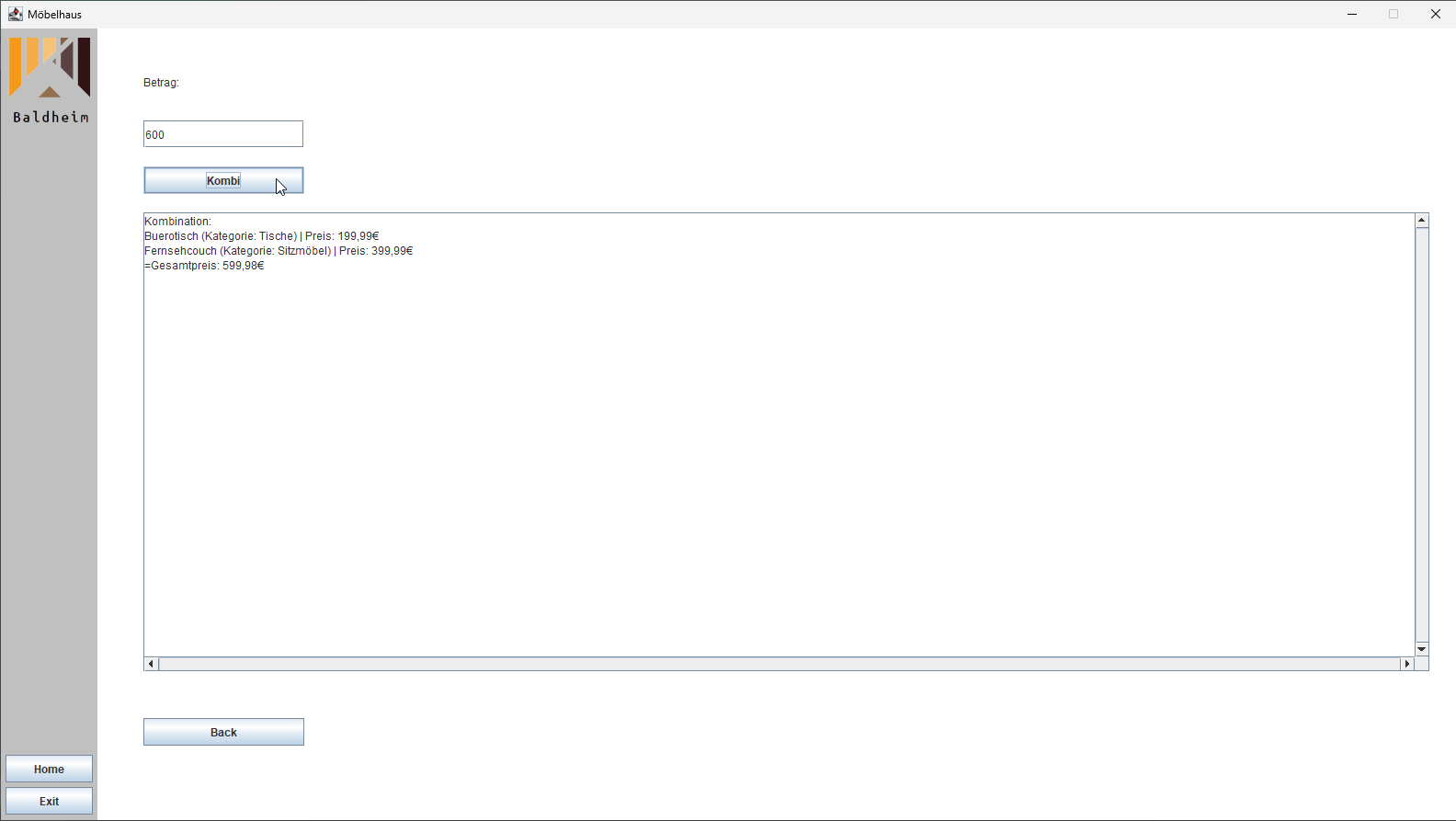


Abbildung 7: Ausgabe einer Kombination an Möbeln für einen eingegebenen Betrag.

Eigene Erstellung.

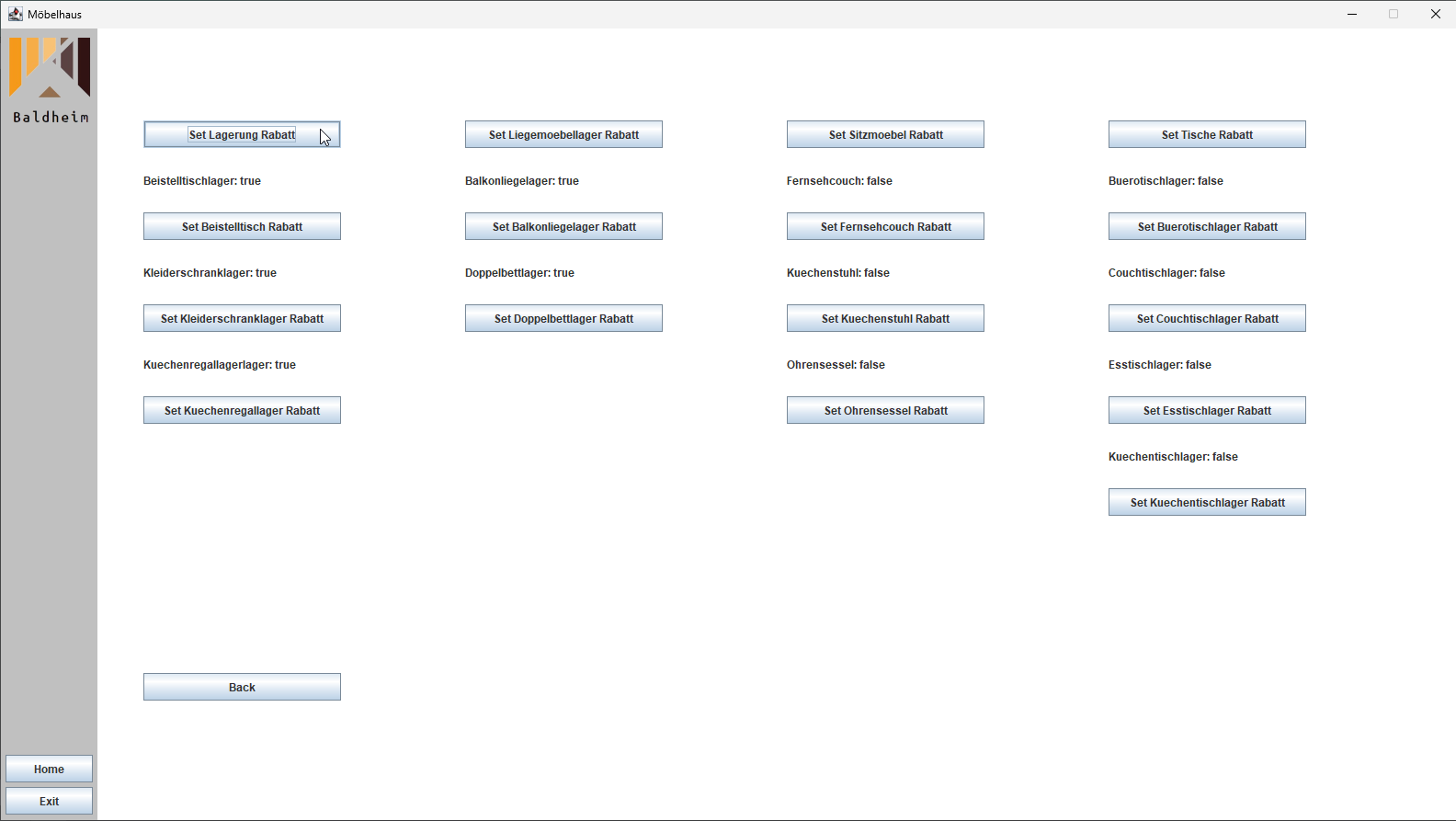


Abbildung 8: Setzen eines Rabattes für eine Kategorie.

Eigene Erstellung.

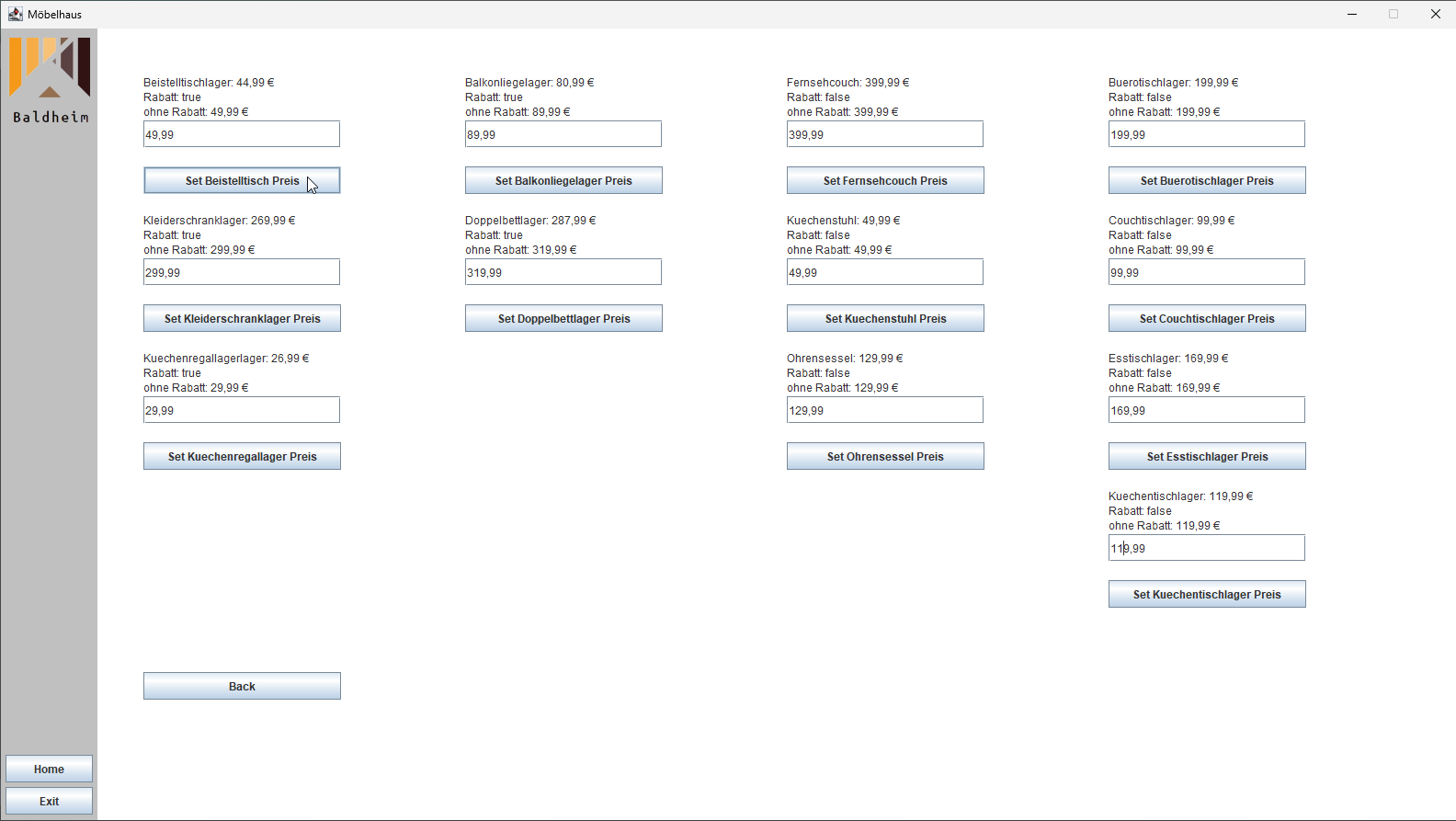
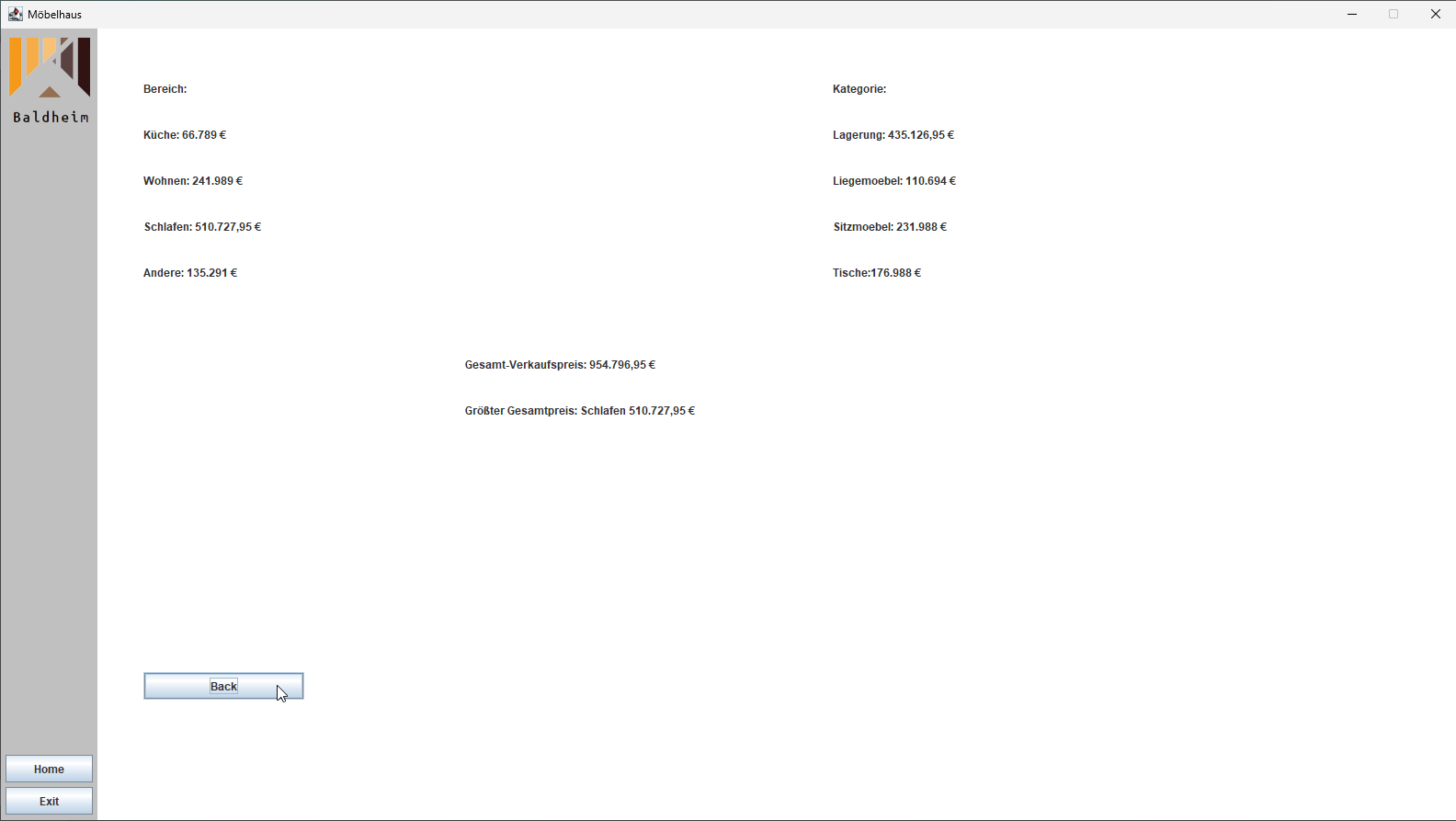


Abbildung 9: Setzen eines Preises für eine Möbelart.

Eigene Erstellung.

Abbildung 10: Ausgabe des Gesamtwertes für Kategorien, Bereiche, Gesamt-Verkaufspreis und der Bereich mit dem größten Gesamtverkaufspreis.

Eigene Erstellung.

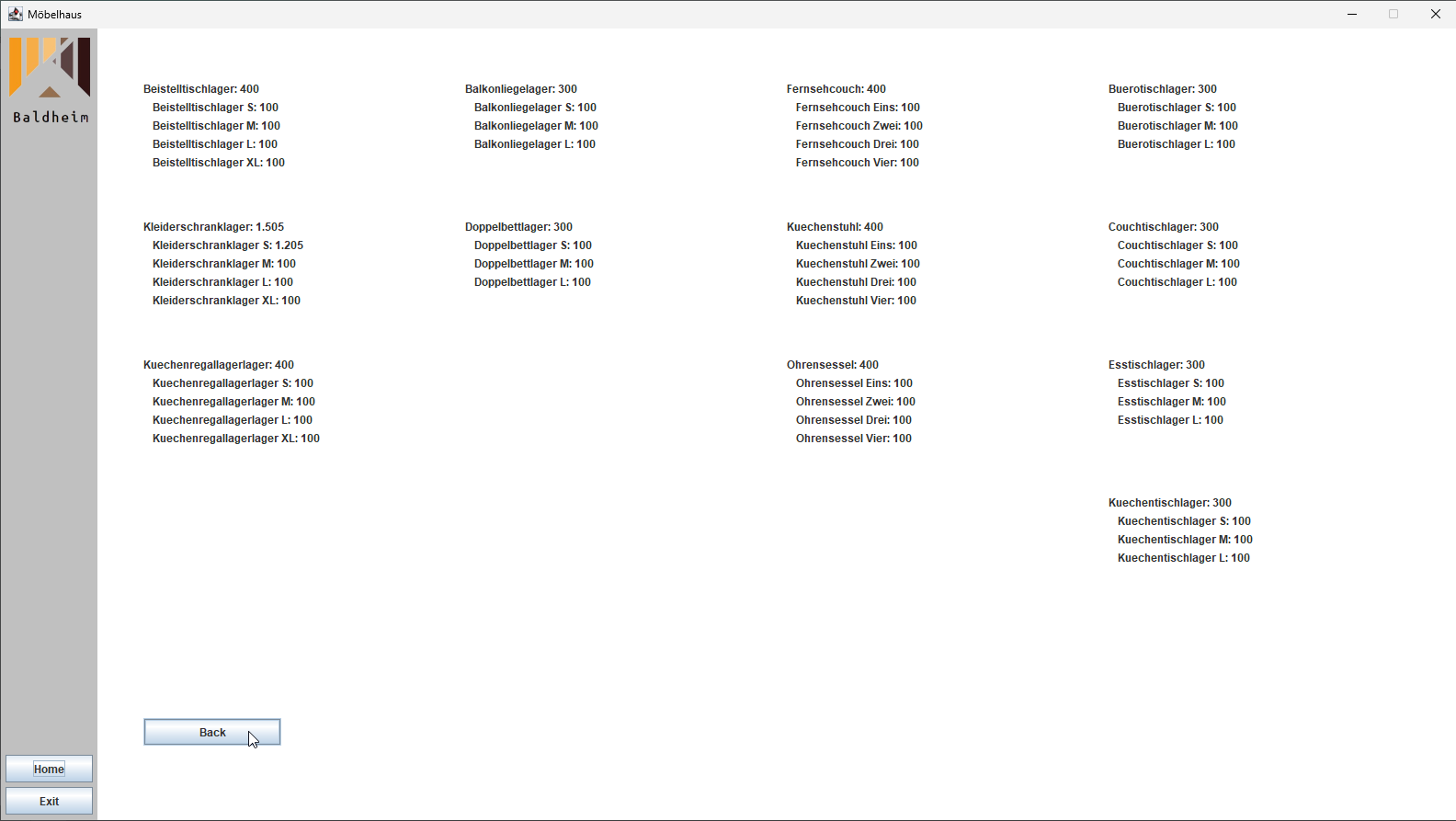
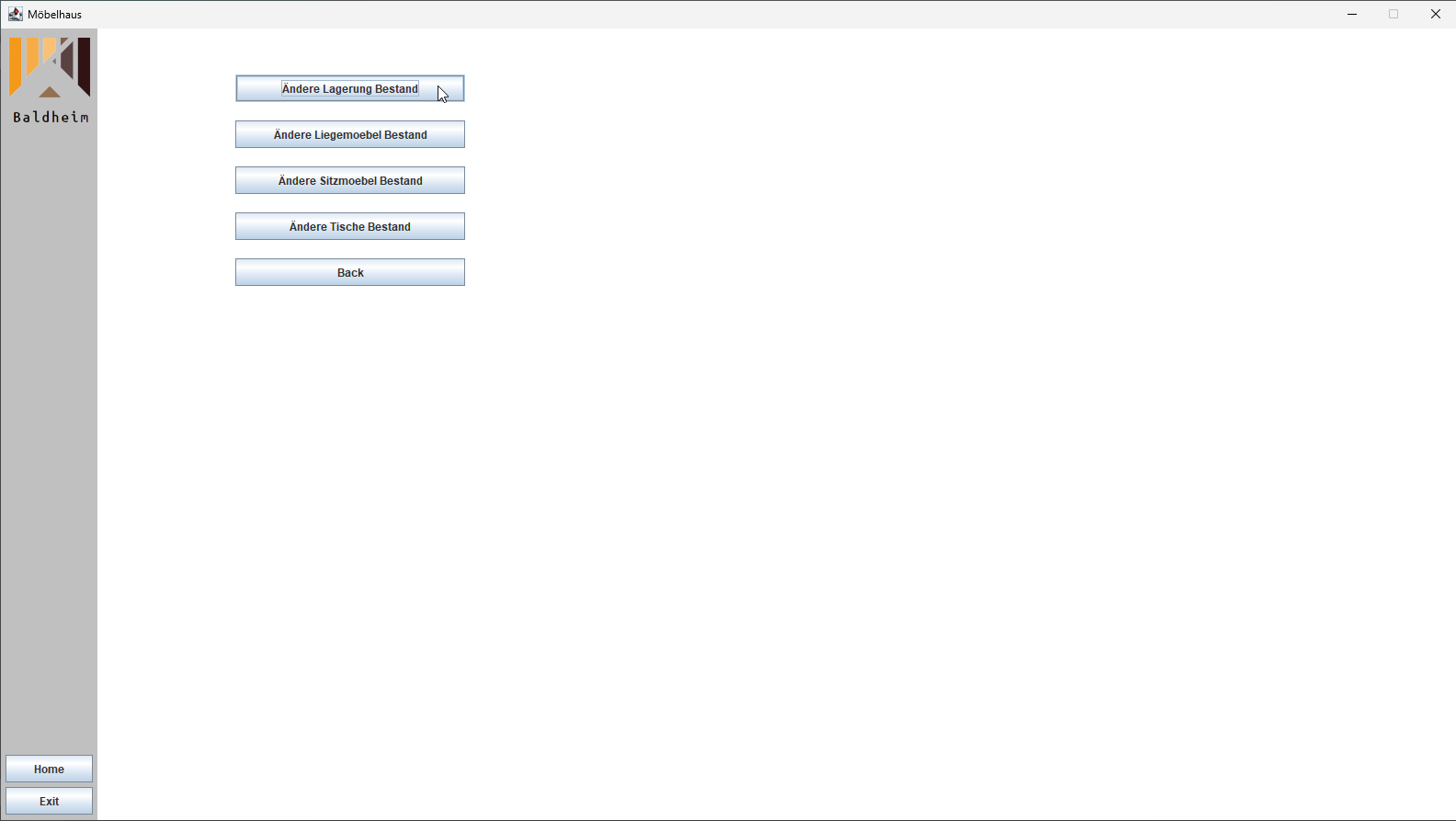


Abbildung 11: Ausgabe des Lagerbestands.

Eigene Erstellung.

Abbildung 12: Änderung des Lagerbestands.

Eigene Erstellung.

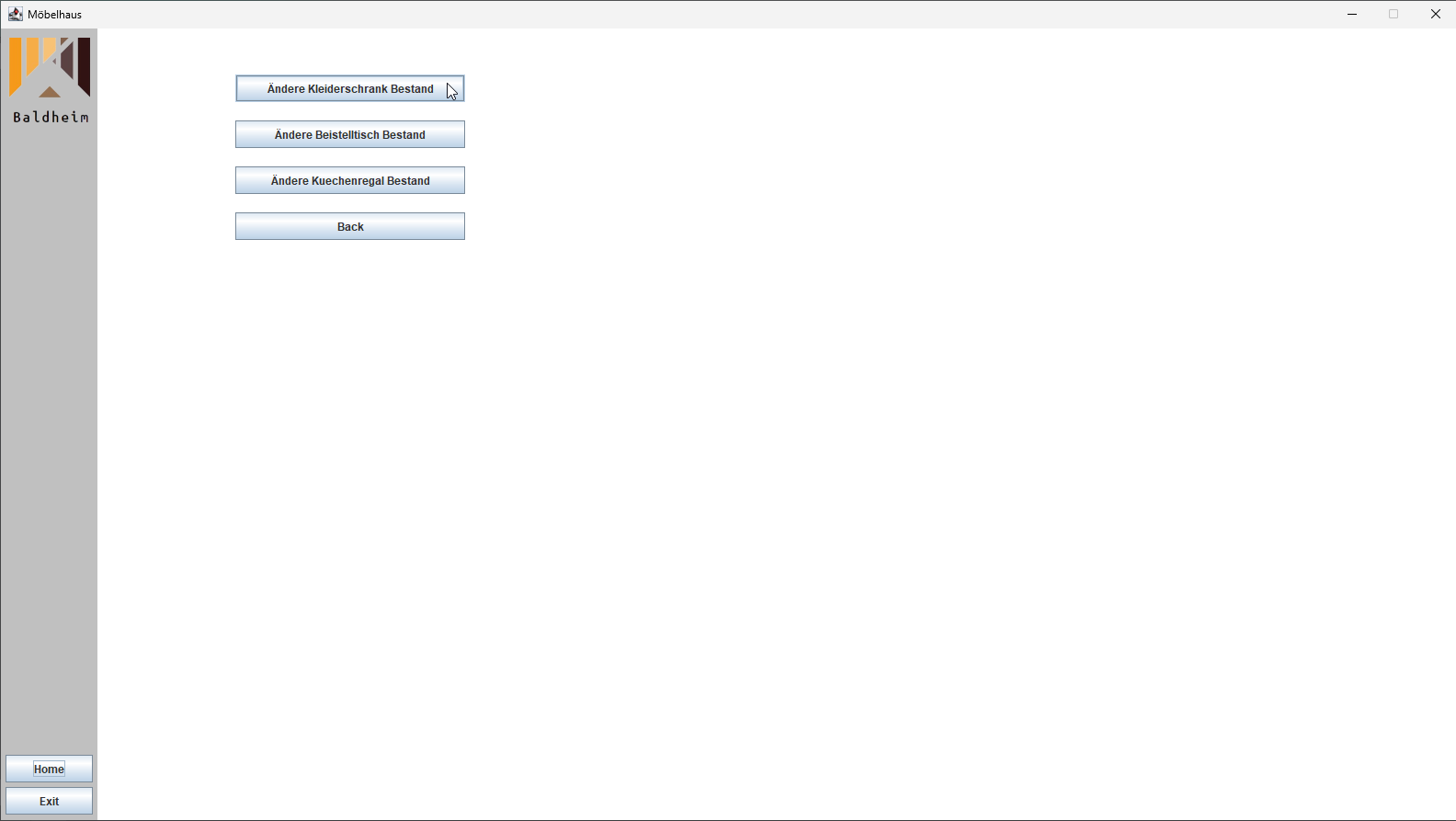
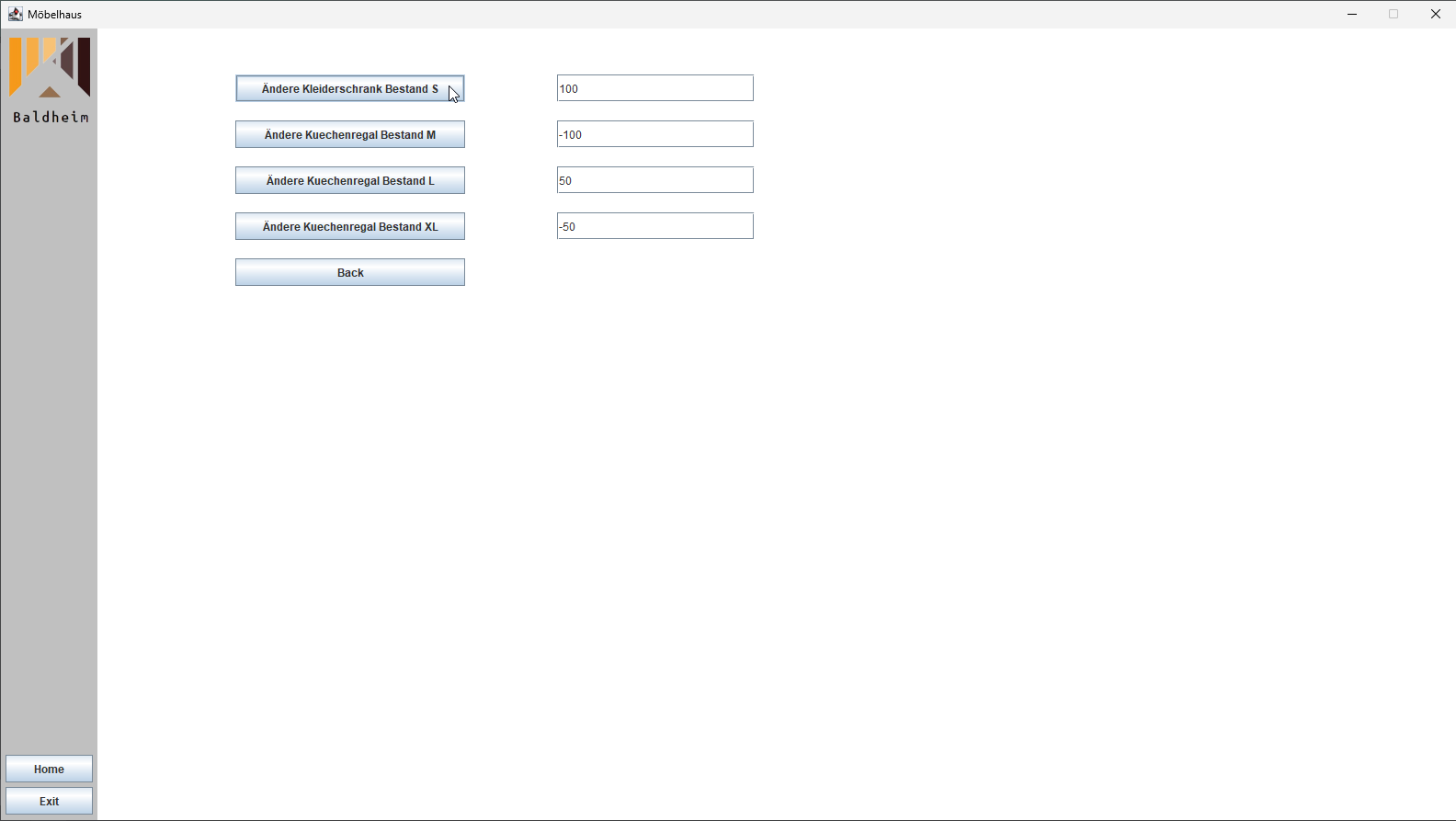


Abbildung 13: Änderung des Lagerbestands.

Eigene Erstellung.

Abbildung 14: Änderung des Lagerbestands.

Eigene Erstellung.

# **1.6 Fazit/Ausblick**

Die Entwicklung von Softwarelösungen erfordert eine sorgfältige Abwägung verschiedener Entscheidungen auf unterschiedlichen Ebenen, beginnend bei abstrakten Konzepten wie der Darstellungstechnik bis hin zu konkreten Implementierungsdetails wie der Wahl von Datentypen. Eine entscheidende Komponente dieses Prozesses ist die Auswahl der Darstellungstechnik, die auf verschiedenen Ebenen betrachtet werden kann, darunter die Benutzeroberfläche.

Im Rahmen dieser Entscheidungsfindung wurden zunächst die Lösungsalternativen für die abstrakte Ebene der Darstellungstechnik untersucht, insbesondere für die grafische Benutzeroberfläche und Konsolenanwendungen. Die GUI ermöglicht eine interaktive Benutzererfahrung durch visuelle Elemente wie Schaltflächen und Menüs. Obwohl GUIs eine intuitivere Interaktion und eine breitere Palette von Funktionen bieten, gehen sie mit höherem Ressourcenverbrauch und Entwicklungskomplexität einher im Vergleich zu Konsolenanwendungen. Innerhalb des GUI-Kontextes wurden die Potenziale von JavaFX und Swing als Darstellungstechnologien erörtert. JavaFX bietet eine modernere und flexible Benutzeroberfläche mit fortgeschrittenen Gestaltungsmöglichkeiten. Swing hingegen zeichnet sich durch seine Stabilität, ausgereiften Funktionen und eine umfassende Dokumentation für Einsteiger aus, was es zu einer verlässlichen Wahl macht. Die Entscheidung für die Wahl zwischen GUI und Konsolenanwendung fiel aufgrund des Fokus auf Benutzerfreundlichkeit und interaktive Erfahrungen zugunsten einer GUI-Anwendung. Dies wurde durch die Auswahl von Swing als GUI-Technologie unterstützt, aufgrund der umfassenderen und einfacheren Dokumentation für Einsteiger. In Bezug auf die Darstellung von Preisen wurde die Verwendung ganzzahliger Zahlen für die Präzision und einfache Vergleichbarkeit gegenüber Fließkommazahlen bevorzugt, obwohl dies zusätzlichen Umrechnungsaufwand mit sich bringt. Die Beschreibung des Programms zeigt eine strukturierte Klassenhierarchie zur Modellierung verschiedener Aspekte der Möbelverwaltung, wobei die Entscheidungen und Begründungen für verschiedene Entwurfsentscheidungen erläutert wurden.

Insgesamt verdeutlicht die schriftliche Ausarbeitung eine fundierte Herangehensweise an die Entwicklung von Softwarelösungen, die sowohl technische Anforderungen als auch Benutzererwartungen berücksichtigt.

Ein wichtiger Punkt in der Programmierung ist die Weiterentwicklung des Programms und das Hinzufügen von zusätzlichen Funktionen. Mit der Sicht auf weitere Funktionen für das Programm besteht die Möglichkeit, den Preis für ein spezielles Möbelstück festzulegen. Dieser wird bisher in der Klasse „Lager“ definiert. Für besondere Möbelstücke, wie zum Beispiel der Verkauf von Ausstellungsstücken oder von leicht beschädigten Möbelstücken, die wie zum Beispiel beim international größten Möbeleinzelhändler „Ikea“ in einer sogenannten Fundgrube erworben werden können, kann ein anderer Preis festgelegt werden. Diese sind auf eine winzige Menge begrenzt und der Preis für diese könnte in der Klasse „Moebelstueck“ mit dem Attribut „Verkaufspreis“ neu definiert werden. Zudem bietet das Programm aufgrund seiner objektorientierten Programmierung eine einfache Möglichkeit zur Erweiterung des Lagersortiments mit zusätzlichen Kategorien, Möbelarten und Bereichen.

# **Anhang**

**Anhangsverzeichnis**

[Anhang 1: Anfang der abstrakten Klasse „Moebelstueck“ V](#_Toc160495299)

[Anhang 2: Anfang der Klasse „Lagerung“ VI](#_Toc160495300)

[Anhang 3: Die finale Klasse „Beistelltischlager“ VII](#_Toc160495301)

[Anhang 4: Anfang der abstrakten Klasse „Lagerunglager“ VII](#_Toc160495302)

[Anhang 5: Methoden zum Lesen und Schreiben eines Objekts aus und in eine Datei der Kategorielager-Klassen, exemplarisch an der Klasse „Lagerunglager“ VIII](#_Toc160495303)

[Anhang 6: Die generische Klasse „Lager“ IX](#_Toc160495304)

[Anhang 7: Anfang der Klasse „Lagersystem“ X](#_Toc160495305)

[Anhang 8: Die Klasse „Moebelhaus“ XI](#_Toc160495306)

Ein Bild, das Text, Screenshot, Schrift, Dokument enthält.

Automatisch generierte Beschreibung

Anhang 1: Anfang der abstrakten Klasse „Moebelstueck“.

Eigene Erstellung.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Anhang 2: Anfang der Klasse „Lagerung“.

Eigene Erstellung.

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

Anhang 3: Die finale Klasse „Beistelltischlager“.

Eigene Erstellung.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Anhang 4: Anfang der abstrakten Klasse „Lagerunglager“.

Eigene Erstellung.

Ein Bild, das Text, Screenshot, Schrift, Dokument enthält.

Automatisch generierte Beschreibung

Anhang 5: Methoden zum Lesen und Schreiben eines Objekts aus und in eine Datei der Kategorielager-Klassen, exemplarisch an der Klasse „Lagerunglager“.

Eigene Erstellung.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Anhang 6: Die generische Klasse „Lager“.

Eigene Erstellung.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Anhang 7: Anfang der Klasse „Lagersystem“.

Eigene Erstellung.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Anhang 8: Die Klasse „Moebelhaus“.

Eigene Erstellung.

# **Literaturverzeichnis**

Minh, N. H. (2019). Java IO ObjectInputStream and ObjectOutputStream Examples. Abgerufen am 15.02.2024 von https://www.codejava.net/java-se/file-io/java-io-objectinputstream-and-objectoutputstream-examples

Goldberg , D. (1991). What Every Computer Scientist Should Know About Floating-Point Arithmetic. California: ACM Computing Surveys, Vol. 23, No 1, S. 6f.

1. Vgl. Goldberg , D. (1991). What Every Computer Scientist Should Know About Floating-Point Arithmetic. California: ACM Computing Surveys, Vol. 23, No 1, S. 6f. [↑](#footnote-ref-1)
2. Vgl. Minh, N. H. (2019). Java IO ObjectInputStream and ObjectOutputStream Examples. Abgerufen am 15.02.2024 von https://www.codejava.net/java-se/file-io/java-io-objectinputstream-and-objectoutputstream-examples [↑](#footnote-ref-2)