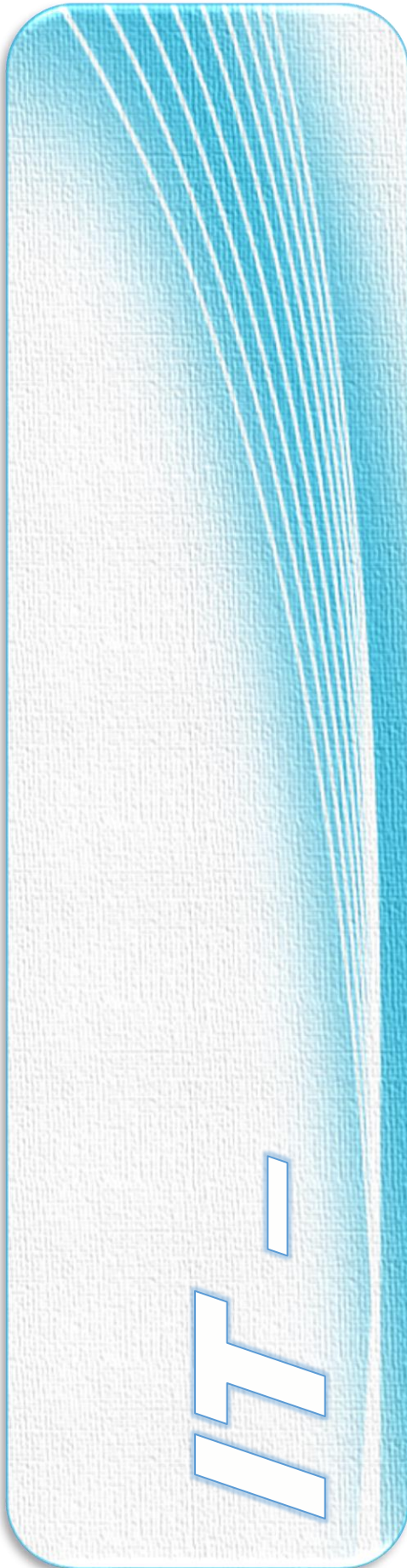




Unternehmen für Bildung.



Funktionalität in Anwendungen realisieren

Modul: ITBS FIA L
Version: 1.0

Januar 2023

Erstellt von: IT-Ausbildungsabteilung

Modul: ITBS FIA L Funktionalität in Anwendungen realisieren

Version, Datum: 1. Januar 2023

Erstellt von: IT-Ausbildungsabteilung

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne Genehmigung des Herausgebers reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Die Modulunterlage und alle ihre Bestandteile wurden mit großer Sorgfalt erstellt und geprüft. Dennoch können Fehler nicht vollkommen ausgeschlossen werden. Der Herausgeber und die Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Literaturhinweis

Titel: IT Berufe Fachstufe II FIAE & FIDPA

ISBN: 987-3-14-220072-9

Verlag: Westermann Gruppe

Inhaltsverzeichnis

Inhaltsverzeichnis	3
1. Modulare Softwarekomponenten planen.....	5
1.1. Das Prinzip der Modularisierung präsentieren	6
1.2. Softwarekomponenten mithilfe von UML beschreiben	8
1.3. Entwurfsmuster unterscheiden und verwenden	19
1.4. Clean-Code-Prinzipien präsentieren	23
2. Datenstrukturen von Softwarekomponenten planen und implementieren	27
2.1. Grundlegende Datenstrukturen beschreiben	27
2.2. Datenstrukturen bedarfsgerecht auswählen.....	33
2.3. Grundlegende Datenstrukturen von Python unterscheiden und einsetzen	36
2.4. Grundlegende Datenstrukturen von Java unterscheiden und einsetzen	40
3. Funktionalität von Softwarekomponenten planen und implementieren	45
3.1. Algorithmen entwerfen und implementieren.....	45
3.2. Rekursive Algorithmen umsetzen	51
3.3. Sortierverfahren beschreiben, implementieren und anwenden	53
3.4. Komprimierungsverfahren beschreiben, implementieren und anwenden.....	57
3.5. Verschlüsselungsverfahren beschreiben, implementieren und anwenden.....	62
3.6. Hashfunktionen beschreiben, implementieren und anwenden	67
3.7. JSON-Dateien im- und exportieren und Webservices nutzen	71
3.8. Funktionalität im Zusammenhang mit Datenbanken realisieren.....	75
4. Funktionalität von Softwarekomponenten testen	82
4.1. Testverfahren auswählen.....	82
Nachteile	82
4.2. Testfälle formulieren	84
4.3. Automatisierte Unittests entwerfen, implementieren und anwenden.....	88
5. Softwarekomponenten und deren Schnittstellen dokumentieren	91



Durch die Entwicklung einer Software sollen u.a. Arbeitsabläufe und Prozesse automatisiert werden. Dazu werden diese im Vorfeld analysiert und aus dieser Analyse die funktionalen und nicht funktionalen Anforderungen an die zukünftige Software abgeleitet. Auch wenn die nicht funktionalen Anforderungen, z. B. Wartbarkeit oder Effizienz, eine große Rolle bei der Erstellung einer Software spielen, steht natürlich zunächst einmal die Umsetzung der funktionalen Anforderungen der Software im Vordergrund. In der Regel wird die Software dazu in verschiedene Module aufgeteilt, welche möglichst unabhängig voneinander sind und über Schnittstellen miteinander kommunizieren.



Bevor die einzelnen Module implementiert werden, werden zunächst die notwendigen Datenstrukturen ausgewählt sowie die einzelnen Komponenten und deren Algorithmen geplant. Dazu wird in der Regel eine Beschreibungssprache wie UML verwendet. Nachdem die einzelnen Module oder Komponenten implementiert wurden, müssen diese noch getestet werden. Dazu werden zunächst Testfälle formuliert und danach mithilfe dieser Testfälle die Softwarekomponenten manuell oder automatisch getestet.

Da die Funktionalität der einzelnen Module nur über entsprechende Schnittstellen genutzt werden kann, sollte auf die sorgfältige Dokumentation dieser Schnittstellen ein besonderes Augenmerk gelegt werden.

Die Inhalte dieses Lernfeldes werden exemplarisch immer wieder an verschiedenen Beispielen verdeutlicht. Grundlage dafür ist das Szenario rund um den Modellkunden „Yachthafen Resort“.

1. Modulare Softwarekomponenten planen

- **Sie sollen sich die Grundlagen der Modularisierung von Software erarbeiten und Softwarekomponenten mithilfe von UML planen. Dabei sollen Sie Entwurfsmuster einsetzen und Clean-Code-Prinzipien berücksichtigen.**

Die Anforderungen, welche an moderne Software gestellt werden, werden immer umfangreicher. Dies betrifft sowohl die funktionalen als auch die nicht funktionalen Anforderungen. Gleichzeitig werden flexible und auf die Bedürfnisse des Kunden optimal abgestimmte Lösungen gefordert. Aus diesem Grund muss für die Software eine optimale Softwarearchitektur gefunden werden. Hierfür gibt es einige Lösungsansätze.

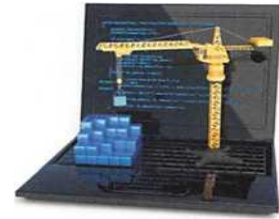
Ein Lösungsansatz ist die Modularisierung der Software, welche in diesem Kapitel näher betrachtet werden soll. Ein wesentlicher Vorteil der Modularisierung besteht in der leichteren Wartbarkeit von einmal geschriebenen Programmen. In diese Richtung zielt auch die Anwendung der Clean-Code-Prinzipien bei der Programmentwicklung. Da bei der Implementierung von Anfang an die Clean-Code-Prinzipien berücksichtigt werden sollten, werden diese an dieser Stelle bereits vorgestellt. Zur Planung von modularen Softwarekomponenten wird heutzutage hauptsächlich die Modellierungssprache UML verwendet. In diesem Bereich werden die vorhandenen Kenntnisse vertieft und das Sequenz- und das Zustandsdiagramm als neue Beschreibungsmittel eingeführt.



1.1. Das Prinzip der Modularisierung präsentieren

➤ Sie sollen sich das Prinzip der Modularisierung bei der Programmentwicklung erarbeiten.

Die Modularisierung ist bei der Entwicklung einer modernen Softwarearchitektur ein wesentlicher Bestandteil. Generell besteht die Idee der Modularisierung darin, ein komplexes Softwaresystem aus einfacheren Einzelbausteinen zusammenzusetzen. Diese einfacheren Bausteine werden auch Module genannt. Je nach Softwareebene können sich die Module auch wiederum aus anderen Modulen zusammensetzen. Die Modularisierung bringt viele Vorteile für die Softwareentwicklung mit sich.



Vorteile der Modularisierung

Erhöhung der Verständlichkeit des Programms

Durch die Aufteilung des Programms in verschiedene Module und Untermodule wird die Verständlichkeit für den Aufbau des Programms und dessen Funktionsweise erhöht.

Gute Wiederverwendbarkeit von Programmteilen

Gleicher Programmcode muss nicht mehrfach geschrieben werden. Stattdessen können einmal entwickelte Module in verschiedenen Programmen wiederverwendet werden. Auf diese Weise lassen sich zukünftige Programme schneller und zuverlässiger entwickeln.

Problemlose Austausch- und Veränderbarkeit von Programmteilen

Bei Beibehaltung der Funktionalität der Modulschnittstellen lassen sich Programmteile des Moduls austauschen oder an neue Anforderungen anpassen, ohne dass der Rest des Programms davon beeinflusst wird.

Begünstigt die flexible und parallele Entwicklung der Software

Programmmodule sollen weitestgehend unabhängig sein und nur über Schnittstellen miteinander interagieren. Wenn die Schnittstellen definiert und bekannt sind, dann ist eine parallele Komponentenentwicklung in großen Softwareteams möglich.

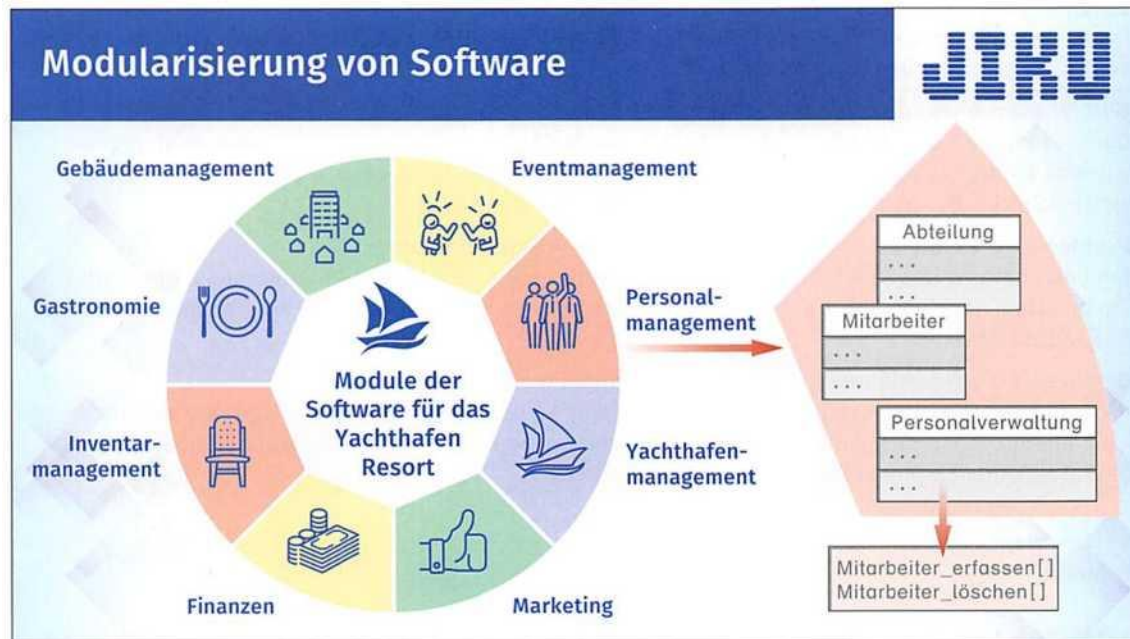
Modul

Ein Modul ist ein Baustein eines Softwaresystems, welcher eine funktional geschlossene Einheit darstellt und einen bestimmten Dienst bereitstellt, welcher über eindeutig spezifizierte Schnittstellen genutzt werden kann

Module können weitere Module wie Funktionen, Klassen usw. enthalten. So ist die Zerlegung der Programmfunktionalität in einer Hierarchie möglich. Eine Modularisierung kann auf verschiedenen Ebenen des Programms erfolgen und unterschiedliche Ausprägungen besitzen.

Modularisierung auf verschiedenen Programmebenen	
Methode/Funktion	Stellt die unterste Ebene der Modularisierung dar. Jede Methode oder Funktion sollte nur eine Aufgabe erfüllen.
Klasse	In der objektorientierten Programmierung stellt die Klasse den Prototypen eines Moduls dar und eignet sich besonders gut zur Modularisierung von Software. Allerdings bedarf es auch hier eines wohlgedachten Klassendesigns, damit die Vorteile der Modularisierung voll zum Tragen kommen.
Softwarekomponente	Softwarekomponenten werden wieder aus verschiedenen Untermodulen, z.B. Klassen, zusammengesetzt und stellen die nächsthöhere Programmebene dar. In Softwarekomponenten werden i.d.R. Basis- Funktionalitäten von Modulen zu fachspezifischen Diensten zusammengefasst.

Als Beispiel ist ein Teil der geplanten Modularisierung der Softwarelösung dargestellt, welche von JIKU IT-Solutions für ein Unternehmen zu entwickeln ist, welches das „Yachthafen Resort“ betreibt und bewirtschaftet. Zum „Yachthafen Resort“ gehören neben dem eigentlichen Yachthafen auch mehrere Gebäude für Übernachtung, Gastronomie, Eventveranstaltungen und ein Wellnessbereich.



Um eine gute Modularisierung zu erreichen, sollten einige Regeln beim Design von Modulen beachtet werden.

Wichtige Designregeln für Module

Modulgeschlossenheit

Ein Modul soll nur für eine Aufgabe zuständig sein und eine in sich geschlossene Einheit darstellen. Die Kommunikation mit dem Modul findet ausschließlich über eine eindeutig spezifizierte Schnittstelle statt. Interne Änderungen am Modul dürfen keine Rückwirkungen auf das übrige System haben. Ein Modul muss sich ohne Kenntnisse über seinen inneren Aufbau problemlos in andere Programmteile integrieren lassen.

Modulbindung

Unter der Modulbindung ist die Summe der internen Beziehungen zwischen den Operationen und den Daten eines Moduls zu verstehen. Ein Modul sollte eine hohe Modulbindung aufweisen.

Modulkopplung

Die Modulkopplung gibt den Grad an, wie stark verschiedene Module miteinander verbunden sind.

Je größer die Modulkopplung ist, desto mehr Schnittstellen gibt es, die beachtet werden müssen.

Dies erhöht die Gefahr von Fehlern.

Minimalität der Schnittstelle

Die Schnittstelle eines Moduls sollte nur so umfangreich wie nötig ausfallen. Je kleiner die Schnittstelle eines Moduls ist, desto geringer ist die Gefahr einer hohen Modulkopplung. Außerdem werden die benötigten Testfälle zur Überprüfung des Systems reduziert.

Testbarkeit

Die Korrektheit eines Moduls muss auch ohne Kenntnis seiner Einbettung in das Gesamtsystem nachprüfbar sein. Modultests in einer vom Gesamtsystem abgekapselten Umgebung sind ein geeigneter Weg zur Durchführung aller nötigen Testfälle.

✓ Kompetenzcheck

1. Geben Sie an, welche Aussagen richtig sind.
 - a. Modularisierung ist ein zentraler Bestandteil moderner Softwareentwicklung.
 - b. Die Wiederverwendbarkeit von Quellcode wird durch Modularisierung gefördert.
 - c. Die Austauschbarkeit von Programmteilen wird durch die Modularisierung erschwert.
 - d. Ein Modul ist eine funktional abgeschlossene Einheit.
 - e. Klassen eignen sich besonders gut zur Modularisierung.
 - f. Eine Modulschnittstelle sollte möglichst umfangreich ausfallen.
 - g. Unter Modulkoppelung versteht man die Summe der internen Beziehungen zwischen den Daten und den

Operationen eines Moduls.

2. Diskutieren Sie in Partnerarbeit eventuelle Nachteile oder Herausforderungen der Modularisierung und präsentieren Sie diese im Anschluss der Klasse.
3. Entwickeln Sie Ideen, wie eine hohe Modulbindung und die Forderung nach einer minimalen Schnittstelle in Bezug auf Klassen umgesetzt werden können und diskutieren Sie diese anschließend im Klassenverband.
4. Bearbeiten Sie im Handout die Lernsituation 1 mit der Aufgabe 1.

1.2. Softwarekomponenten mithilfe von UML beschreiben

- **Sie sollen Ihre Kenntnisse über das Anwendungsfall-, das Klassen- und das Aktivitätsdiagramm wiederholen und vertiefen. Weiterhin erarbeiten Sie sich neues Wissen über Zustands- und Sequenzdiagramme und wenden diese an.**

Beim Entwurf und der Dokumentation von Software kommen die unterschiedlichsten Modellierungssprachen zum Einsatz. Zum Beispiel wird UML zur Modellierung von objektorientierten Programmen eingesetzt oder das ERM zum Entwurf des Datenmodells für relationale Datenbanken.

Die Grundlagen für UML wurden bereits im zweiten Ausbildungsjahr gelegt. Gegenstand der Betrachtungen waren dort das Anwendungsfall-, das Klassen- und das Aktivitätsdiagramm. Diese werden in diesem Kapitel noch mal wiederholt und gefestigt. Des Weiteren werden das Zustands- und Sequenzdiagramm neu eingeführt.



Als Grundlage dient wieder ein Fallbeispiel aus dem Projekt „Yachthafen Resort“. In diesem Beispiel sollen auf der Basis einer einfachen Anforderungsspezifikation mithilfe von UML-Diagrammen verschiedene Aspekte einer Softwarekomponente entworfen und modelliert werden.

Softwareprojekt „Yachthafen Resort“
Liegeplatzverwaltung



Im Rahmen des Projektes soll eine Softwarekomponente für die Vermietung von Liegeplätzen entwickelt werden. Diese gehört zum Modul des Yachthafenmanagements.

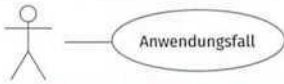
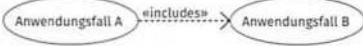
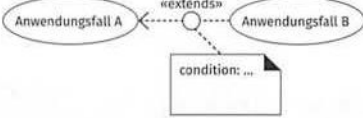
Folgende Anforderungen an die Funktionalität werden gestellt:

- Kunden sollen einen Liegeplatz mieten und kündigen können.
- Kunden sollen sich im Yachthafen mit ihrer Yacht an- und abmelden können.
- Mitarbeiter des Yachthafens sollen in der Lage sein, einzelne Liegeplätze für Wartungsarbeiten zu sperren.
- Alle Nutzer des Programms sollen sich die aktuelle Liegeplatzbelegung ansehen können.

Um eine hohe Softwarequalität des Programms zu gewährleisten, hat sich das Projektteam dazu entschieden, das Programm objektorientiert zu entwickeln und die moderne Modellierungssprache UML zur Planung zu verwenden.



(1) Anwendungsfalldiagramm

Übersicht über wichtige Notationselemente des Anwendungsfalldiagramms		
Notationselement	Bezeichnung	Beschreibung
	Systemgrenze	Die Systemgrenze umfasst alle Anwendungsfälle, welche das System bereitstellt. Für diese Anwendungsfälle muss später das System die entsprechenden Funktionen bereitstellen.
	Anwender oder Akteur	Hier wird beschrieben, welche externen Benutzer das System verwenden. Dabei ist es auch möglich, dass das System mit anderen externen Systemen interagiert. In diesem Fall wird ein Rechteck statt dem Strichmännchen verwendet.
	Anwendungsfall	Beschreibt eine Aktion oder eine Anzahl von Aktionen, die von dem System für die Benutzer bereitgestellt werden.
	Assoziation	Mit einer Assoziation (Strich) wird eine Beziehung zwischen den Akteuren und den Anwendungsfällen dargestellt.
	Include-Beziehung	Durch eine Include-Beziehung wird die unbedingte Einbindung der Funktionalität eines Anwendungsfalls in einen anderen Anwendungsfall dargestellt. In dem Beispiel schließt der Anwendungsfall A den Anwendungsfall B ein, d. h., wenn A ausgeführt wird, muss auch B ausgeführt werden.
	Extend-Beziehung	Durch eine Extend-Beziehung wird die bedingte Einbindung der Funktionalität eines Anwendungsfalls in einen anderen Anwendungsfall dargestellt. In dem Beispiel kann der Anwendungsfall A durch den Anwendungsfall B erweitert werden, d. h., wenn A ausgeführt wird, kann unter bestimmten Bedingungen auch B ausgeführt werden.
	Generalisierung/Spezialisierung	Eine Generalisierung/Spezialisierung zeigt die Beziehung zwischen einem allgemeinen und einem spezialisierten Element. Eine Generalisierung/Spezialisierung kann sowohl zwischen Anwendungsfällen als auch zwischen Akteuren dargestellt werden.



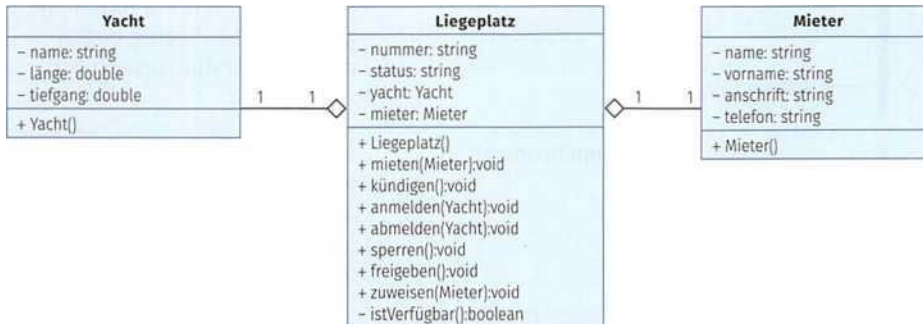
Dieses Anwendungsfalldiagramm kann folgendermaßen gelesen werden: Ein Kunde des „Yachthafen Resorts“ soll einen Liegeplatz mieten und kündigen können. Außerdem muss er in der Lage sein, über das System seine Yacht im Hafen an- und abzumelden, wenn er seinen gemieteten Liegeplatz aktiv nutzt. Den Mitarbeitern des Yachthafens soll es wiederum zur Wartungszwecken ermöglicht werden, über das System Liegeplätze zu sperren und wieder freizugeben und notfalls einem Mieter einen Ausweichliegeplatz zuzuweisen. Bevor ein Liegeplatz gemietet oder als Ausweichliegeplatz zugewiesen werden kann, muss immer vom System überprüft werden, ob der entsprechende Liegeplatz auch verfügbar ist. Alle Benutzer des Systems sollen sich zusätzlich über die aktuelle Belegung des Yachthafens jederzeit informieren können. Die Vermietung und Bewirtschaftung von Liegeplätzen wurden hier natürlich vereinfacht dargestellt. Für eine komplexere Aufgabenstellung würde auch das Anwendungsfalldiagramm komplexer ausfallen. Alternativ können auch mehrere Anwendungsfalldiagramme erstellt werden.

(2) Klassendiagramm

Übersicht über wichtige Notationselemente des Klassendiagramms								
Notationselement	Bezeichnung	Beschreibung						
Klassenname Eigenschaften (Attribute) Methoden (Funktionen)	Klasse	Klassen werden in der UML durch Rechtecke dargestellt, die den Namen der Klasse und/oder die Eigenschaften und Methoden der Klasse enthalten. Klassenname, Attribute und Methoden werden durch eine horizontale Linie getrennt. Der Klassenname steht im Singular und beginnt mit einem Großbuchstaben. Eigenschaften können näher beschrieben werden, z. B. durch ihren Typ. Methoden können ebenfalls durch Parameter usw. zusätzlich beschrieben werden.						
Klassenname Klassenname	Alternative Klassendarstellungen	Bei den alternativen Klassendarstellungen werden meistens die Eigenschaften und Methoden weggelassen. Sie dienen vor allem dazu, die Beziehungen zwischen den einzelnen Klassen zu modellieren.						
+ , - , # , ~ + schwimmen(): void	Sichtbarkeit für Eigenschaften und Methoden	<table><tr><td>+</td><td>public</td></tr><tr><td>-</td><td>private</td></tr><tr><td>#</td><td>protected</td></tr></table> Mit der Sichtbarkeit werden die Zugriffsmodifizier der OOP beschrieben.	+	public	-	private	#	protected
+	public							
-	private							
#	protected							
Klasse A Klasse B	Assoziation gerichtete Assoziation	Eine Assoziation ist eine Beziehung zwischen zwei Klassen. Es ist keine Teil-Ganzes-Beziehung. Gerichtete Assoziationen werden durch einen zusätzlichen Pfeil gekennzeichnet.						
Klasse A Klasse B	Aggregation	Eine Aggregation ist eine Beziehung zwischen zwei Klassen, wobei es sich um eine Teil-Ganzes-Beziehung handelt. Die Aggregation wird durch eine Verbindungslinie zwischen den Klassen mit einer nicht ausgefüllten Raute am Ende zur „Ganzes“-Klasse beschrieben.						
Klasse A Klasse B	Komposition	Eine Komposition ist eine Beziehung zwischen zwei Klassen, wobei es sich um eine „starke“ Teil-Ganzes-Beziehung handelt. Die Komposition wird durch eine Verbindungslinie zwischen den Klassen mit einer ausgefüllten Raute am Ende zur „Ganzes“-Klasse beschrieben.						
Klasse A 1 0..* Klasse B	Multiplizität	Die Multiplizität gibt an, wie viele Objekte der Klasse A mit wie vielen Objekten der Klasse B in Beziehung stehen können. Die Angaben können Assoziationen, Aggregationen oder Kompositionen zugeordnet werden.						
Klasse A Klasse B Interface A Klasse A	Vererbung	In der UML wird die Vererbung durch eine Linie mit einer dreieckigen, nicht ausgefüllten Pfeilspitze dargestellt, die von der Unterklasse zur Oberklasse weist. Wird von einem Interface geerbt, dann wird anstatt einer durchgehenden Linie eine Strichlinie verwendet.						

Umsetzung eines Klassendiagramms für das Beispiel

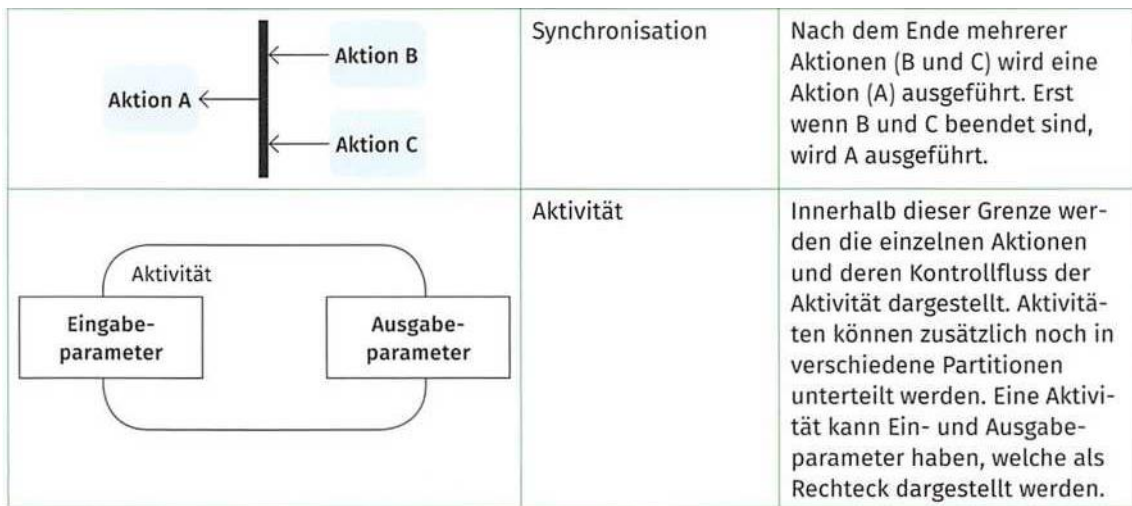
In einem Klassendiagramm für das Fallbeispiel könnten die Klassen „Yacht“, „Mieter“ und „Liegeplatz“ mit ihren Eigenschaften, Methoden und den entsprechenden Sichtbarkeiten genau beschrieben und die Beziehungen zwischen diesen dargestellt werden.



Auf die Angabe der Get- und Set-Methoden wurde bei allen Klassen bewusst verzichtet und nur die Arbeitsmethoden dargestellt. Auch hier ist wieder nur eine vereinfachte Variante umgesetzt worden. Es ist mit Sicherheit davon auszugehen, dass bei den einzelnen Klassen noch weitere Eigenschaften und Methoden für komplexere Aufgabenstellungen geplant werden müssen. Die Klassen „Liegeplatz“ und „Mieter“ bzw. „Liegeplatz“ und „Yacht“ sind als Aggregation miteinander verbunden. In beiden Fällen ist der Liegeplatz die Aggregationsklasse. Diese Beziehung wurde gewählt, weil jeweils ein Mieter und eine Yacht zeitweise ein Bestandteil des Liegeplatzes sein können. Auf der anderen Seite können aber auch alle entsprechenden Objekte unabhängig voneinander im Programm existieren. Auch in diesem Fall ist durchaus vorstellbar, dass die bestehenden Beziehungen erweitert werden, wenn zum Beispiel neue Klassen hinzukommen. Dieses könnte u.a. der Fall sein, wenn für die Klasse „Liegeplatz“ oder die Klasse „Yacht“ neue Unterklassen (Spezialisierungen) durch Vererbung gebildet werden. In diesem Fall müssten eventuell auch einige Sichtbarkeiten von private (-) auf protected (#) geändert werden. Die Methoden der Klasse „Liegeplatz“ wurden direkt aus dem Anwendungsfalldiagramm abgeleitet.

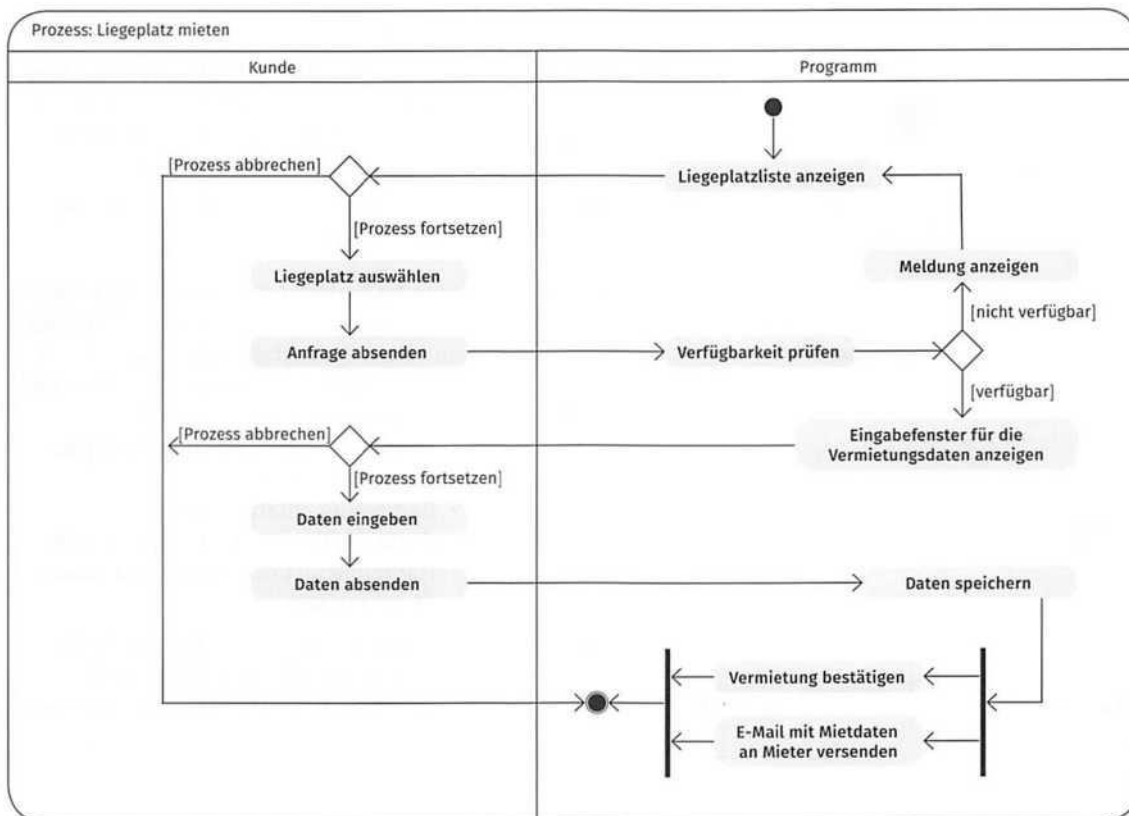
(3) Aktivitätsdiagramm

Übersicht über wichtige Notationselemente des Aktivitätsdiagramms		
Notationselement	Bezeichnung	Beschreibung
	Startknoten Endknoten Ablaufende	Der Startknoten markiert den Beginn eines Ablaufes. Der Endknoten markiert das Ende eines Ablaufes. Das Ablaufende markiert das Ende eines Zweiges.
Aktion	Aktion	Sie beschreibt das Verhalten, welches eine Veränderung herbeiführt.
	Kante	Sie dient zur Angabe des Kontrollflusses zwischen den einzelnen Aktionen.
	Verzweigung	Je nachdem, ob die Bedingung A oder die Bedingung B erfüllt ist, wird die Aktion A oder die Aktion B ausgeführt.
	Zusammenführung	Nach dem Ende der Aktion A oder der Aktion B wird die Aktion C ausgeführt.
	Aufspaltung	Nach dem Ende einer Aktion (A) werden mehrere Aktionen (B und C) gleichzeitig begonnen und parallel ausgeführt.



Umsetzung eines Aktivitätsdiagramms für das Beispiel

In dem abgebildeten Aktivitätsdiagramm für das Fallbeispiel wird der Prozess zum Mieten eines Liegeplatzes modelliert. In diesem Fall ist es nur ein Beispiel für eine Vielzahl von Aktivitätsdiagrammen, welche für die Aufgabenstellung erstellt werden könnten.



Der Prozess als solcher ist in zwei Partitionen aufgeteilt. In der einen Partition werden die Aktionen des Kunden und in der anderen die Aktionen des Programms abgebildet. Das Programm stellt dabei den Benutzern entsprechende Eingabemaschinen zur Verfügung, durch die die Benutzer auch die Möglichkeit erhalten, den Prozess abbrechen. Nach Auswahl eines Liegeplatzes wird, wie schon im Anwendungsfalldiagramm geplant, zunächst dessen Verfügbarkeit überprüft. Nach erfolgreicher Überprüfung werden die notwendigen Daten erfasst und gespeichert. Danach wird dem Kunden der erfolgreiche Abschluss des Mietprozesses im Programm angezeigt und parallel dazu eine E-Mail mit allen relevanten Daten an ihn versendet. Erst wenn diese beiden Aktionen vom Programm abgeschlossen sind, wird auch die gesamte Aktivität als abgeschlossen betrachtet.

(4) Zustandsdiagramm

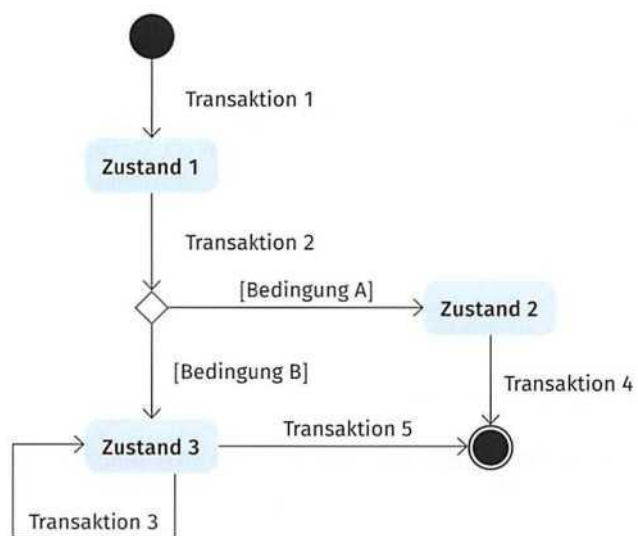
Das Zustandsdiagramm dient dazu, die Zustände zu dokumentieren, welche ein Objekt annehmen kann. Somit wird das dynamische Verhalten eines Systems beschrieben. Im Gegensatz zu den Aktivitätsdiagrammen, mit deren Hilfe

die Aktionen eines Systems abgebildet werden, konzentriert man sich beim Zustandsdiagramm auf die Beschreibung der Reaktionen des Systems. Zustandsdiagramme eignen sich dadurch beispielsweise sehr gut zur Modellierung des Verhaltens von Benutzeroberflächen, weil diese in den meisten Fällen keine eigenen Aktionen durchführen, sondern auf die Benutzereingaben reagieren. Eingesetzt werden Zustandsdiagramme in der Analyse- und Entwurfsphase, meistens als Ergänzung von Klassendiagrammen. Dabei illustriert das Zustandsdiagramm die verschiedenen Zustände, welche ein Objekt annehmen kann, und die Funktionen, welche die Zustandsänderungen bewirken. Dafür stehen im Zustandsdiagramm nur wenige Elemente zur Darstellung zur Verfügung.

Übersicht über wichtige Notationselemente des Zustandsdiagramms		
Notationselement	Bezeichnung	Beschreibung
	Anfangszustand	Besonderer Zustandstyp eines Objektes, welche keine eingehende Transaktion besitzt. Es darf nur einen einzigen Anfangszustand geben.
	Endzustand	Besonderer Zustandstyp eines Objektes, welcher die dargestellte Ebene von Zuständen terminiert. Danach erfolgt keine Zustandsänderung mehr.
	Zustand	Benennt den Zustand, den ein Objekt einnehmen kann.
	Transaktion	Eine Transaktion stellt einen Statuswechsel dar, welcher von einem Quellzustand zu einem Zielzustand führt. Dabei ist es auch möglich, dass der Quellzustand und der Zielzustand identisch sind. • Ereignis (trigger), das den Zustandsübergang auslöst • Bedingung (guard), die beim Zustandsübergang erfüllt sein muss • Handlung (action), die den Zustandsübergang bewirkt
	Verzweigung	Je nachdem, ob die Bedingung A oder die Bedingung B erfüllt ist, wird der Zustand A oder der Zustand B angenommen.

Das zentrale Element des Zustandsdiagramms sind die Zustände eines Objektes. Ein Zustand bleibt für die Zeitspanne zwischen zwei Ereignissen bestehen und ist durch einen eindeutigen Namen definiert. Start- und Endzustand eines Objektes bilden spezielle Zustände, da zu einem Startzustand kein Übergang stattfindet und dem Endzustand keine Zustandsänderung mehr folgt. Ansonsten wird der Übergang von einem Zustand in einen anderen durch eine Transaktion ausgelöst.

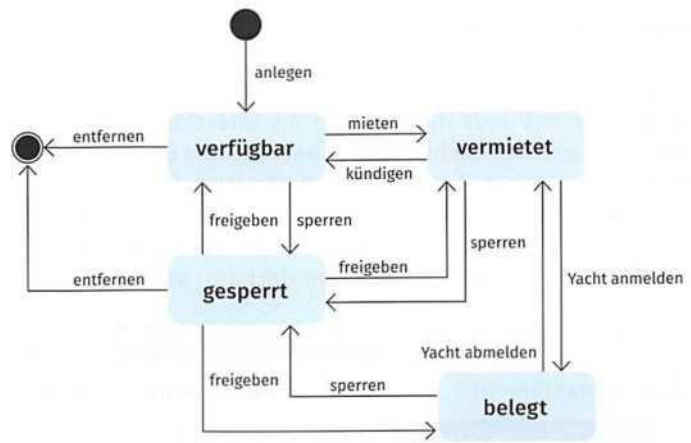
Transaktionen können als Ereignisse oder als Handlungen beschrieben werden. Außerdem kann jede Transaktion mit Bedingungen versehen werden, welche beschreiben, wann ein Übergang erfolgt. Ergen sich aus einer Transaktion bei unterschiedlichen Bedingungen unterschiedliche Zustände, dann wird das durch eine Verzweigung realisiert. Transaktionen können auch auf den Ausgangszustand zurückführen.



Umsetzung eines Zustandsdiagramms für das Beispiel

In einem Zustandsdiagramm für das Fallbeispiel werden die einzelnen Zustände dargestellt, welche ein Liegeplatz im

Programm haben kann. Diese werden als aktueller Status auch in dem entsprechenden Liegeplatzobjekt vermerkt. Weiterhin enthält dieses Diagramm auch Informationen über die Transaktionen, durch welche die Zustandsänderungen ausgelöst werden. Diese entsprechen überwiegend den Anwendungsfällen, welche im Anwendungsfalldiagramm geplant wurden. Die Transaktionen spiegeln sich auch in den Methoden der Klasse „Liegeplatz“ wider.



Weitere Beispielaufgabe zum Zustandsdiagramm

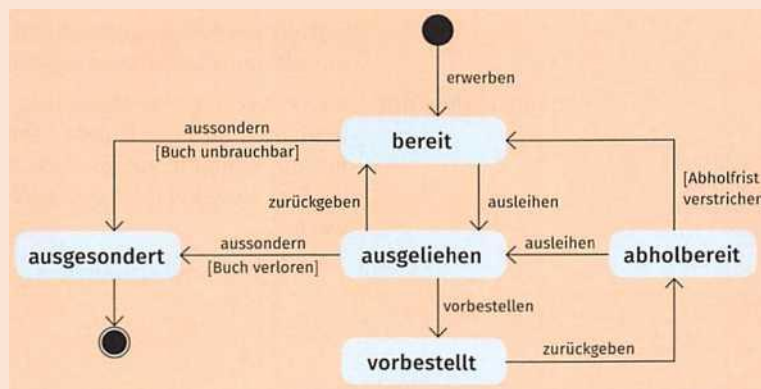
Beispielaufgabe

Die JIKU IT-Solutions soll ein Programm zur Verwaltung von Büchern in einer Bibliothek entwickeln. Für die weitere Entwicklung und als Diskussionsgrundlage soll dokumentiert werden, welchen Status ein Buch innerhalb der Bibliothek annehmen kann. Folgende Fakten sind bekannt:

- Ein Buch wird von der Bibliothek erworben und danach zur Ausleihe bereitgestellt (Status bereit).
- Wird ein Buch ausgeliehen, dann wird dieses entsprechend im Programm vermerkt, bis es wieder zurückgegeben wurde.
- Ein ausgeliehenes Buch kann vorbestellt werden, was wiederum zu einem Eintrag im Programm führt.
- Wird ein vorbestelltes Buch zurückgegeben, dann ist es für eine gewisse Zeitspanne abholbereit und kann vom Vorbesteller ausgeliehen werden. Wird dieser Zeitraum überschritten, dann wird es wieder zur allgemeinen Ausleihe zur Verfügung gestellt.
- Ein Buch wird ausgesondert (steht somit nicht mehr für den Ausleihprozess zur Verfügung), wenn es entweder seitens der Bibliothek als unbrauchbar eingestuft wird oder es bei einem Ausleihvorgang verloren ging. Dieser Status des Buches wird auch in dem Programm vermerkt.

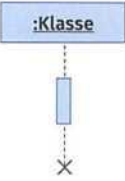
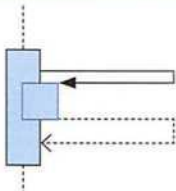
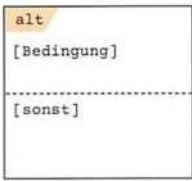
Aus dieser Beschreibung soll ein Zustandsdiagramm entwickelt werden, welches alle Zustände und deren Übergänge abbildet.

Lösung

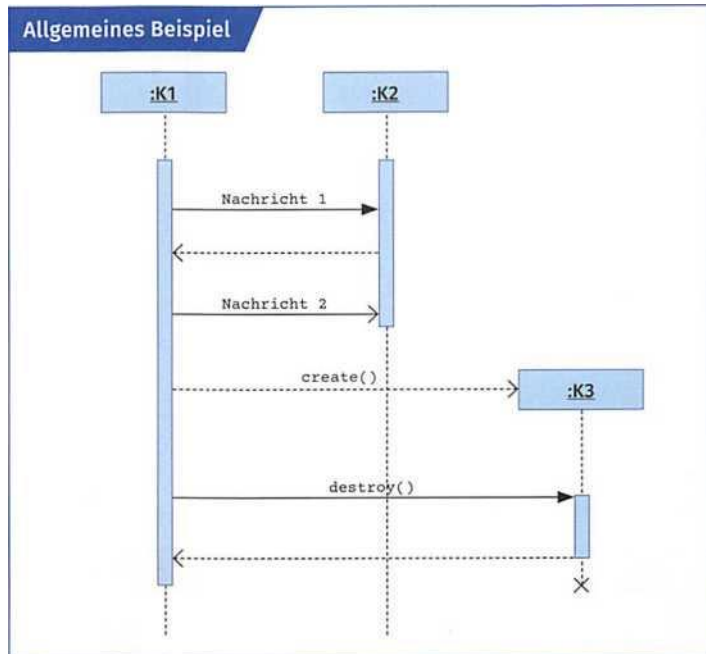


(5) Sequenzdiagramm

Sequenzdiagramme werden eingesetzt, um den Ablauf interaktiver, ereignisgesteuerter Systeme zu verdeutlichen. Mit ihrer Hilfe werden die Interaktionen zwischen den Objekten modelliert. Man erkennt, wann (bei welcher Nachricht oder bei welchem Ereignis) ein Objekt aktiviert wird und mit welchen Nachrichten es reagiert. Auch Sequenzdiagramme finden in verschiedenen Phasen des Softwareentwicklungsprozesses ihren Einsatz. So werden beispielsweise in der Analysephase Sequenzdiagramme verwendet, um den Nachrichtenfluss der abzubildenden Prozesse und in der Entwurfsphase die Interaktion von Benutzern mit der Benutzeroberfläche zu beschreiben.

Übersicht über wichtige Notationselemente des Sequenzdiagramms		
Notationselement	Bezeichnung	Beschreibung
	Interaktionsrahmen	Der Interaktionsrahmen umschließt die Verhaltensdefinitionen des zu beschreibenden Systems. Links oben wird der Name des Interaktionsdiagramms notiert.
	Objekt Lebenslinie Aktivierung Löschung	Objekt: Rechteck, Beschriftung: Klassenname Lebenslinie: senkrechte, gestrichelte Linie; Zeit, in der das Objekt existiert Aktivierung: schmales Rechteck, Zeit, in der Objekt aktiv ist, z. B. Ausführung einer Methode Löschung: Die Löschung eines Objektes wird durch ein X auf der Lebenslinie gekennzeichnet.
	Synchrone Nachricht	Bei synchronen Nachrichten wartet das System auf eine Antwort vom Empfänger, bevor es seine Prozesse fortsetzt. Synchrone Nachrichten werden als Pfeil mit einer gefüllten Pfeilspitze dargestellt.
	Asynchrone Nachricht	Bei asynchronen Nachrichten wartet das System nicht auf eine Antwort des Empfängers, sondern setzt seine Prozesse ohne Unterbrechung fort. Asynchrone Nachrichten werden als Pfeil mit durchgehender Linie und offener Pfeilspitze dargestellt.
	Antwort	Antworten werden von dem Empfänger zurück an den Absender gesendet, nachdem die Operation zu einem Ergebnis gekommen ist. Antworten werden als gestrichelte Linie und Pfeil mit offener Pfeilspitze dargestellt.
	Selbstdelegation	Ein Objekt ruft eine Methode auf, die es selbst implementiert. Das Objekt kann ggf. erst fortfahren, wenn die aufgerufene Methode die Verarbeitung beendet und eine Antwort gesendet hat.
	Interaktionsfragment	Interaktionsfragmente werden verwendet, um Abläufe zu beeinflussen. Damit können Schleifen oder Alternativen realisiert werden. Hierfür gibt es unterschiedliche Begriffe, welche links oben ins Fragment eingetragen werden. Wichtige Begriffe sind z. B.: • alt (Alternative) • loop (Schleife) • opt (Option) • par (Parallel)

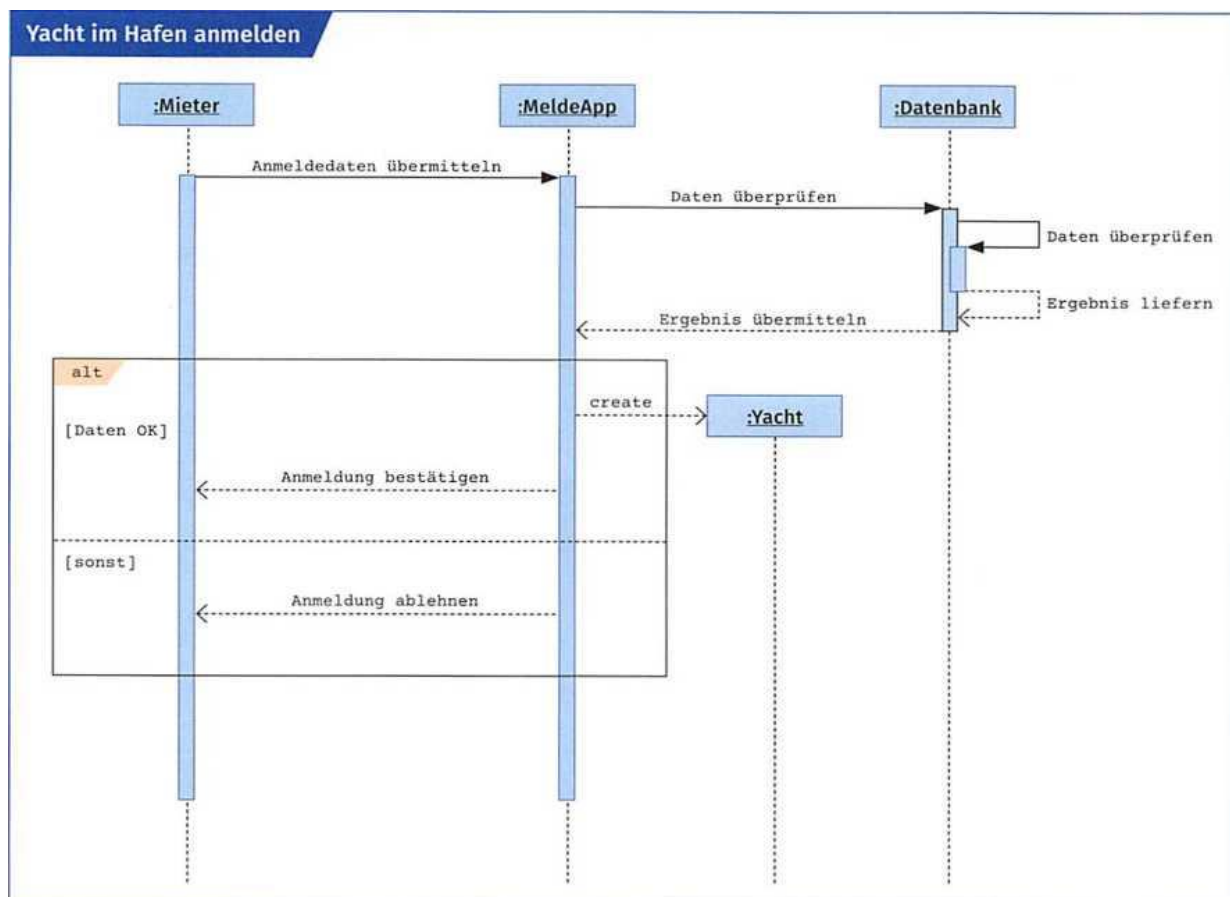
Das Sequenzdiagramm wird durch einen Interaktionsrahmen umschlossen und durch einen treffenden Namen beschrieben. Die konkret handelnden Objekte werden im Modell durch den Namen ihrer Klasse ersetzt und durch Rechtecke visualisiert. Von diesen Rechtecken gehen dann die senkrechten Lebenslinien aus. Dabei verläuft eine gedachte Zeitlinie senkrecht von oben nach unten. Die Nachrichten, welche zwischen den Objekten ausgetauscht werden, werden durch waagerechte Pfeile zwischen den Objekt-Lebenslinien beschrieben. Auf diesen Pfeilen werden die Nachrichtennamen notiert und sie werden entsprechend ihrem zeitlichen Ablauf eingetragen. Durch einen entsprechenden Befehl kann ein neues Objekt erzeugt werden. Der Pfeil der Nachricht mündet in diesem Fall direkt in dem Rechteck für das neue Objekt. Die Zerstörung eines Objektes wird durch ein X angezeigt.



Umsetzung eines Sequenzdiagramms für das Beispiel

In einem exemplarischen Sequenzdiagramm wird der Prozess der Anmeldung einer Yacht im Yachthafen beschrieben. Dabei gibt der Mieter zunächst die erforderlichen Anmeldedaten, z. B. die Daten der Yacht, in die App zur Anmeldung im Yachthafen ein. Diese überprüft die Daten mithilfe einer Datenbank und wartet auf das Prüfergebn. Je nachdem, ob die Anmeldedaten „OK“ sind oder nicht, werden weitere Prozessschritte durch die App ausgeführt. Sind die Daten „OK“, dann wird dem Mieter die Anmeldung durch eine Nachricht bestätigt und ein entsprechendes Objekt der Klasse „Yacht“ im Programm angelegt und mit den Anmeldedaten initialisiert. Anderenfalls erhält der Mieter die Nachricht, dass die Anmeldung abgelehnt wurde.

Auch in diesem Fall wurde der Prozess der Anmeldung sehr vereinfacht dargestellt. Für ein entsprechendes Programm sind mit Sicherheit mehrere Sequenzdiagramme erforderlich.



Weitere Beispielaufgabe zum Sequenzdiagramm

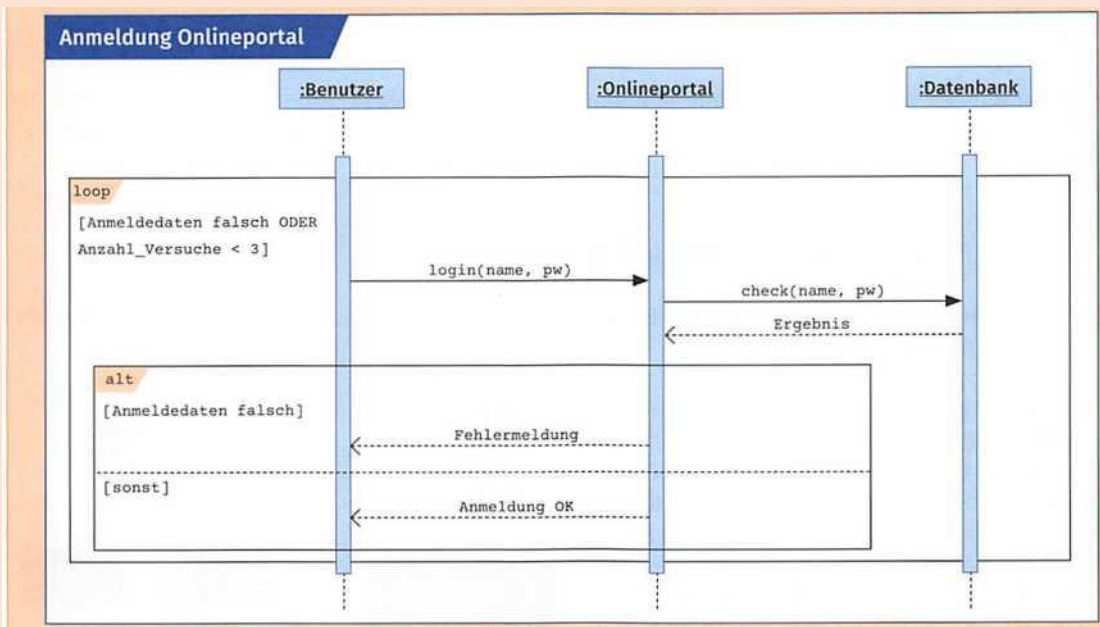
Beispielaufgabe

Im Rahmen der Entwicklung einer Lernsoftware wird von J1KU IT-Solutions u.a. ein Login für das Onlinelernportal entwickelt.

Der Login-Vorgang gestaltet sich folgendermaßen: Benutzer geben ihren Namen und ihr Passwort ein. Die Angaben werden vom Onlineportal an eine Datenbank zur Überprüfung weitergesendet. (Hinweis: Die Überprüfung in der Datenbank selbst muss nicht dargestellt werden.) Die Datenbank sendet das Ergebnis der Überprüfung an das Onlineportal zurück. Für den Fall, dass die Anmeldedaten falsch sind, kann dieser Vorgang noch zweimal wiederholt werden. Ist die Anmeldung dreimal fehlgeschlagen, wird eine entsprechende Fehlermeldung ausgegeben. Ansonsten erhalten die Benutzer die Meldung, dass sie sich ordnungsgemäß eingeloggt haben.

Der beschriebene Ablauf soll in einem Sequenzdiagramm dargestellt werden.

Lösung



✓ Kompetenzcheck

- Geben Sie an, welche Aussagen richtig sind.
 - Aggregation und Komposition sind Bestandteile des Anwendungsfalldiagramms.
 - Im Aktivitätsdiagramm können keine Prozesse parallel ablaufen.
 - Das Zustandsdiagramm zählt zu den Verhaltensdiagrammen.
 - Im Gegensatz zu einem Aktivitätsdiagramm werden mit einem Zustandsdiagramm die Reaktionen eines Objektes abgebildet.
 - Durch eine Transaktion findet im Zustandsdiagramm immer eine Zustandsänderung statt.
 - Transaktionen in Zustandsdiagrammen können mit Bedingungen versehen werden.
 - Ein Sequenzdiagramm bildet das statische Verhalten eines Systems ab.
 - Mit einem Sequenzdiagramm werden die Interaktionen zwischen einzelnen Objekten modelliert, i) In einem Sequenzdiagramm wird zwischen synchronen und asynchronen Nachrichten unterschieden.
 - Ein Objekt in einem Sequenzdiagramm kann sich nicht selbst eine Nachricht senden.
 - Sequenzdiagramme ermöglichen die Darstellung der Erzeugung von Objekten durch andere Objekte.
- Entwickeln Sie ein Zustandsdiagramm für die einzelnen Zustände einer Datei aus der folgenden „vereinfachten“ Beschreibung:

Nach dem Anlegen einer Datei muss diese zum Lesen bzw. Schreiben geöffnet werden. Soll in eine Datei etwas geschrieben werden, dann erhält sie den Status „entsperrt“. Beim Lesen gilt sie nur als geöffnet. Das Lesen und Schreiben einer Datei kann so lange wiederholt werden, bis sie geschlossen wird. Dabei bleibt ihr aktueller Status bestehen. Eine geschlossene Datei kann auch gelöscht werden.

- Es soll der Prozess des Erzeugens, des Schreibens und des Sendens einer E-Mail beschrieben werden: Ein Benutzer legt zunächst mithilfe des Programms eine neue E-Mail-Datei an. Über diesen Vorgang erhält er vom Programm eine Nachricht. Danach schreibt er einen Text und andere notwendige Angaben in diese

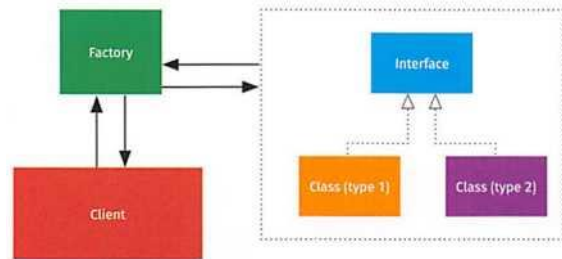
Mail und beauftragt dann im Anschluss das Programm, die E-Mail-Datei an den Empfänger zu versenden. Mit dem Versand wird das entsprechende E-Mail-Objekt automatisch von dem Programm gelöscht und der Benutzer über den Versand informiert.
Bilden Sie diesen Prozess mithilfe eines Sequenzdiagramms ab.

4. Entwickeln Sie weitere verschiedene UML-Diagramme, indem Sie im Handout die Aufgaben 2-6 der Lernsituation 1 bearbeiten. Nutzen Sie die umfangreichen Aufgaben auch als Prüfungsvorbereitung.

1.3. Entwurfsmuster unterscheiden und verwenden

- **Sie sollen sich die Grundlagen über Entwurfsmuster erarbeiten und einige wichtige Entwurfsmuster präsentieren.**

Bei der Planung und dem Entwurf der Architektur von Software kommen zunächst die sogenannten Architekturmuster zum Einsatz. Darüber hinaus kann bei der Detailplanung auf sogenannte Entwurfsmuster zurückgegriffen werden. Während die Architekturmuster den „Stil“ der Gesamtarchitektur beschreiben, dienen Entwurfsmuster dazu, Teilaspekte der Gesamtlösung genauer zu beschreiben. Dabei stellen sie eine Art Lösung für ein Problem dar, welches immer wieder auftritt. Diese Lösung ist ausgereift und hat sich in der Praxis über die Jahre hinweg bewährt. Obwohl die Entwurfsmuster ein konkretes Problem lösen, sind sie flexibel gestaltet, sodass sie an verschiedene Anforderungen anpassbar bleiben.



Entwurfsmuster (Design Pattern)

Ein Entwurfsmuster ist eine anpassbare Lösung für ein wiederkehrendes Problem bei der Entwicklung von Software.

Kategorien von Entwurfsmustern	
Erzeugungsmuster (Creational Design Pattern)	Erzeugungsmuster beschäftigen sich mit dem Klassendesign und der Erstellung bzw. Instanziierung von Objekten. <i>Beispiele:</i> Factory, Singleton, Builder
Strukturmuster (Structural Design Pattern)	Strukturmuster definieren das Zusammenwirken und die Beziehungen von Klassen oder Objekten. <i>Beispiele:</i> Adapter, Bridge, Facade
Verhaltensmuster (Behavioral Design Pattern)	Verhaltensmuster definieren, was eine bestimmte Klasse oder ein Objekt tun soll. Hierunter fallen z. B. das Versenden von Nachrichten oder Aufrufe zwischen Objekten. <i>Beispiele:</i> Observer, Command, Iterator

Der Einsatz von Entwurfsmustern ermöglicht es, Software flexibler und wartbarer zu gestalten. Da diese Muster in der Praxis erprobt sind, können sich Entwickler sicher sein, dass sie mit den Entwurfsmustern eine brauchbare Lösung für ihr Problem erhalten.

Entwurfsmuster sollten nur gezielt zur Lösung von konkreten Problemen eingesetzt werden. Wenn Entwurfsmuster verwendet werden, obwohl gar kein Problem existiert, welches damit gelöst werden soll, kann das in einigen Fällen nicht nur überflüssig, sondern sogar schädlich sein, wenn z.B. das Singleton-Pattern zu viel verwendet wird.

Im Zusammenhang mit den Entwurfsmustern taucht auch immer wieder der Begriff „Anti-Pattern“ auf. Während unter einem Entwurfsmuster (Pattern) eine Lösung für ein Problem verstanden wird, welches positiv für die zu entwickelnde Software ist, werden als Anti-Pattern Lösungen und Verhaltensweisen bezeichnet, welche ungünstig oder sogar schädlich für die zu entwickelnde Software sind. Anti-Pattern werden in verschiedene Bereiche, wie z.B. Architektur- bzw. Entwurfs-Anti-Pattern oder Programmierungs- Anti-Pattern, unterteilt. Ein typisches Architektur- bzw. Entwurfs-Anti-Pattern ist z.B. das „Gottobjekt“. In der objektorientierten Programmierung wird darunter eine Klasse verstanden, welche zu groß ist und zu viel Funktionalität aufweist. Verschiedenste fachliche Aspekte des Softwaresystems sind in dieser Klasse vereint und das Prinzip der Modularisierung wird durchbrochen. Anti-Pattern im Bereich der Programmierung sind z.B. die „Zwiebel“ oder „Copy and Paste“. Bei der „Zwiebel“ wird neue Funktionalität um

alte Funktionalität herumgebaut. Dieses tritt vor allem bei der Erweiterung von Programmen auf und kann über die Jahre zu ganzen „Zwiebelsystemen“ führen, welche am Ende weder zu testen noch zu warten sind. Beim „Copy and Paste“ werden bestehende Code-Passagen herauskopiert und in den eigenen Code eingefügt. Dabei besteht immer die Gefahr, dass Fehler mit kopiert werden. Außerdem ist die Wahrscheinlichkeit groß, dass der Entwickler nicht wirklich den Code versteht, den er auf diese Weise übernimmt.

In einigen Fällen werden auch Entwurfsmuster zu Anti-Pattern, z. B. das Singleton Pattern. Auf der einen Seite wird sichergestellt, dass nur ein Objekt aus einer Klasse erzeugt wird. Gleichzeitig wird dadurch aber die Skalierbarkeit der Anwendung eingeschränkt. Deswegen sollten die Vor- und Nachteile der einzelnen Entwurfsmuster sorgfältig gegeneinander abgewogen werden. Nachfolgend sollen exemplarisch einige Entwurfsmuster vorgestellt werden.

(1) Singleton Pattern

Problem

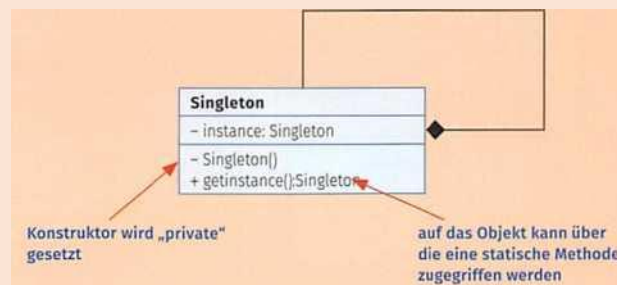
Es soll sichergestellt werden, dass von einer Klasse genau ein Objekt erzeugt wird. Oder anders ausgedrückt: Es soll verhindert werden, dass von einer Klasse mehrere Objekte erstellt werden können. Dieses kann immer dann notwendig sein, wenn z.B. auf zentrale Hardware-Ressourcen wie eine Tastatur, ein Drucker oder ein Scanner zugegriffen werden soll. Des Weiteren kann es auch eine gute Idee sein, nur ein Objekt zu erzeugen, wenn die Instanziierung des Objektes eine gewisse Zeit in Anspruch nimmt, z. B. wenn eine umfangreiche Datei eingelesen werden muss.

Lösung

Das Singleton Pattern gehört zur Kategorie der Erzeugungsmuster, ist sehr einfach und trotzdem eines der mächtigsten Entwurfsmuster.

Das beschriebene Problem wird bei diesem Pattern so gelöst, dass die Singleton-Klasse selbst nur Objekte erzeugen kann und diese auch verwaltet.

Somit kann die Klasse verhindern, nachdem schon ein Objekt existiert, dass weitere Objekte erstellt werden. Das einzelne Objekt kann dann durch eine statische Methode (in den meisten Fällen getInstance()) direkt über die Singleton-Klasse angesprochen werden. Der Konstruktor der Singleton-Klasse wird „private“ gesetzt, sodass er nicht von außen aufgerufen werden kann. Somit kann von außen auch kein Objekt von dieser Klasse erzeugt werden.



Vor- und Nachteile des Singleton Pattern	
Vorteile	Nachteile
• schnell und einfach umzusetzen • eine Möglichkeit, globale Variablen zu vermeiden • kann durch Vererbung spezialisiert werden • Objekt wird im Gegensatz zu statischen Klassen nur dann erzeugt, wenn es im Programm auch wirklich benötigt wird	• zweckentfremdeter Einsatz kann zu „quasi-globalen“ Variablen führen (Variablen, welche zwar in einer Klasse gekapselt sind, aber wie globale Variablen im Programm behandelt werden und damit den objektorientierten Ansatz untergraben) • bei Systemen mit parallelen Abläufen (Threads) muss sichergestellt werden, dass auch wirklich nur eine Instanz existiert • das Testen eines Singleton ist mitunter schwierig oder sogar unmöglich • der Zeitpunkt der Freigabe von Ressourcen (z. B. das Schließen einer Log-Datei), welche vom Singleton genutzt worden ist, ist teilweise schwer festzulegen

(2) Factory Pattern

Problem

- Es sollen Objekte verwendet werden, welche zur Entwicklungszeit noch nicht bekannt sind.
- Der Erstellungsprozess von Objekten soll einfach und unkompliziert erweiterbar sein.
- Es soll der Erstellungsprozess von Objekten von deren Verwendung getrennt werden.

Typische Anwendungsfälle sind daher Frameworks oder Klassenbibliotheken, die als Grundgerüst für die Entwicklung moderner Applikationen quasi unverzichtbar geworden sind.

Lösung

Das Factory Pattern gehört zu den Erzeugungsmustern. Hierbei wird der Code für die Erstellung eines Objektes vom Verarbeitungscode getrennt. Dazu wird das Objekt (Produkt) in einer eigenen Klasse (Erzeuger, Creator, Factory) erstellt. Der Erzeuger ist zunächst mal abstrakt und delegiert die konkrete Objektinstanziierung wiederum an eine seiner Unterklassen. Erst die Unterklasse entscheidet, welches konkrete Produkt erstellt wird. Da der Client nur die

Abstraktionsklassen verwendet (sowohl beim Erzeuger als auch beim Produkt), ist er vollkommen von den Implementierungen entkoppelt und unabhängig.



Vor- und Nachteile des Factory Pattern	
Vorteile	Nachteile
• gute Erweiterbarkeit um neue Produkte • Erstellungsprozess der Objekte ist von seiner Verwendung entkoppelt • einfache Erweiterbarkeit der Erstellung von Objekten eines konkreten Produktes • Verwendung von Objekten, die zur Entwicklungszeit noch nicht implementiert sind	• Enge Koppelung eines konkreten Erzeugers an ein konkretes Produkt • Für jedes neue Produkt muss ein neuer Erzeuger geschrieben werden, was zu einer hohen Anzahl an Klassen führen kann

(3) Adapter Pattern

Problem

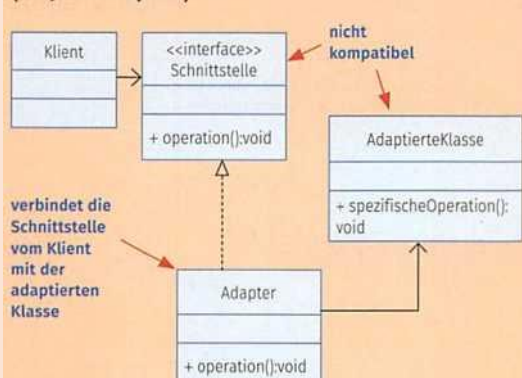
Eine vorhandene wiederverwendbare Komponente hat eine Schnittstelle, welche nicht der benötigten Schnittstelle entspricht, und der Code der Komponente kann nicht geändert werden. Dieses ist z.B. dann der Fall, wenn Klassen von Drittanbietern mit entsprechenden Schnittstellen zur Verfügung stehen, diese aber nicht mit dem eigentlichen Programm kompatibel sind.

Lösung

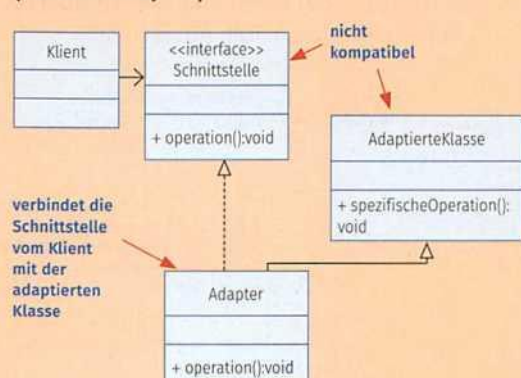
Das Adapter Pattern gehört in die Kategorie der Strukturmuster. Es dient dazu, die Kommunikation von Klassen mit zueinander inkompatiblen Schnittstellen zu ermöglichen. Das Prinzip dieses Musters ist leicht zu verstehen, da es viele verschiedene Adapter gibt, z. B. ein USB-zu-Ethernet-Adapter. Hier liegen zwei Schnittstellen (Ethernet-Schnittstelle, USB-Schnittstelle) vor, welche nicht kompatibel miteinander sind.

Damit beide Schnittstellen miteinander kommunizieren können, wird ein entsprechender Adapter verwendet. Dieses lässt sich auch auf die Software übertragen. Der Klient verwendet weiterhin nur seine Schnittstelle. Die Adapterklasse konvertiert die Schnittstelle einer adaptierten Klasse in die Schnittstelle vom Klient. Dieses lässt sich auf zwei Arten umsetzen: zum einen mittels Delegation (Objektadapter) und zum anderen mittels Vererbung (Klassenadapter).

Umsetzung mittels Delegation (Objektadapter)



Umsetzung mittels Vererbung (Klassenadapter)



Vor- und Nachteile des Adapter Pattern	
Vorteile	Nachteile
• generell erhöhen beide Varianten die Wiederverwendbarkeit von Softwaremodulen • Klassenadapter können optimal an eine Zielklasse angepasst werden • Objektadapter können auch Unterklassen anpassen	• zusätzliche Zwischenklasse • Klassenadapter können nicht zur Anpassung von Unterklassen verwendet werden

(4) Observer Pattern

Problem

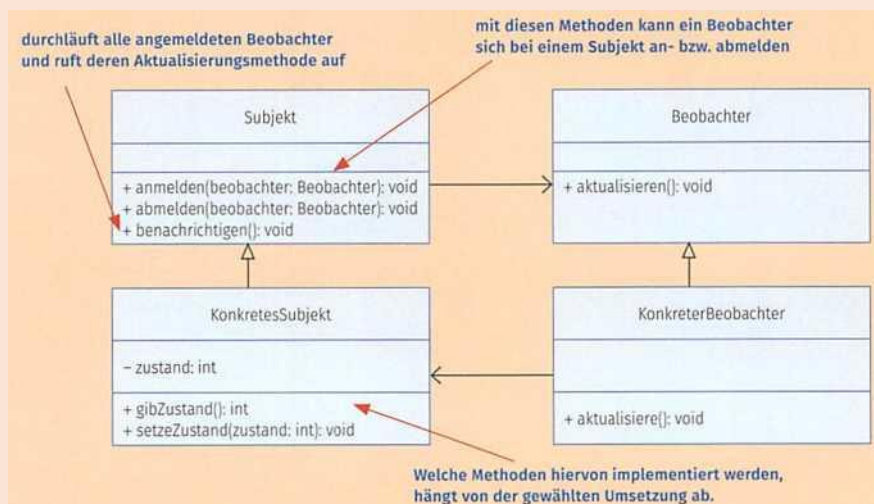
Die Erzeugung und die Verwaltung von Informationen soll von der Informationsverarbeitung und -darstellung getrennt werden. Trotzdem sollen die Informationen bei Änderung in allen Teilen des Programms immer aktuell sein. Dazu müssen die entsprechenden Programmteile miteinander kommunizieren, d. h., bei Änderungen müssen alle davon betroffenen Programmteile über die Änderungen informiert werden. Als Beispiel sei das MVC-Pattern angeführt. Bei Änderung der Daten im „Model“ muss die „View“ darüber informiert werden. Diese holt sich dann die aktuellen Daten vom Model.

Lösung

Das Observer Pattern zählt zu den Verhaltensmustern. Hierbei bietet der Erzeuger der Information (Subjekt) Methoden an, mit deren Hilfe sich Beobachter an- und abmelden können. Findet eine Veränderung statt, „informiert“ das Subjekt alle angemeldeten Beobachter über die Veränderung. Danach werden die Daten bei den Beobachtern aktualisiert.

Hier gibt es drei verschiedene Möglichkeiten der Umsetzung:

- Push Notification: Bei Änderungen im Subjekt werden alle Beobachter benachrichtigt. Die Beobachter müssen sich nach dem Eintreffen der Nachricht die Änderungen selbst vom beobachteten Subjekt holen.
- Push-Update Notification: Bei Änderungen im Subjekt werden alle Beobachter benachrichtigt. Zusätzlich leitet das beobachtete Subjekt alle Änderungen an die Beobachter weiter.
- Pull Notification: Die Beobachter werden nicht benachrichtigt, sondern fragen selbstständig den Zustand des beobachteten Subjekts ab.



Vor- und Nachteile des Observer Pattern	
Vorteile	Nachteile
• Subjekt- und Beobachterklassen können unabhängig voneinander variiert werden • Systeme sind flexibel zu gestalten • neue Beobachter können schnell hinzugefügt werden • die Daten im gesamten System bleiben immer konsistent	• bei größeren Systemen kann es schnell zu Aktualisierungskaskaden kommen • beim Löschen von Beobachtern müssen diese auch beim Subjekt abgemeldet werden • es werden alle Änderungen an den Beobachter weitergeleitet, auch wenn diese für den konkreten Beobachter nicht relevant sind

✓ Kompetenzcheck

- Geben Sie an, welche Aussagen richtig sind.
 - Ein Entwurfsmuster ist eine Lösung für ein wiederkehrendes Problem bei der Softwareentwicklung.
 - Ein Anti-Pattern ist das genaue Gegenstück eines Entwurfsmusters.
 - Es gibt Erzeugungs-, Struktur- und Verhaltensmuster.
 - Die Bridge ist ein Erzeugungsmuster.
 - Mit dem Singleton Pattern soll sichergestellt werden, dass nur ein Objekt aus einer Klasse erzeugt werden kann.
 - Das Factory Pattern wird zu den Strukturmustern gezählt.
 - Beim Factory Pattern wird die Erzeugung von Objekten gekapselt.
 - Durch das Adapter Pattern wird eine Verbindung zwischen zwei inkompatiblen Schnittstellen hergestellt.
 - Das Adapter Pattern kann u.a. durch Mehrfachvererbung realisiert werden.
 - Das Observer Pattern ist ein Verhaltensmuster.
 - Das Observer Pattern kann u.a. im MVC-Architekturmuster verwendet werden.
- Bearbeiten Sie im Handout die Lernsituation 1 mit der Aufgabe 7.

1.4. Clean-Code-Prinzipien präsentieren

- **Sie sollen die grundlegenden Konzepte vom Clean Code verstehen und diese bei der Programm-erstellung anwenden können.**

Clean Code ist ein Begriff aus der Softwaretechnik. Die Idee hinter Clean Code ist, den Quellcode so zu schreiben, dass er leicht verständlich und nachvollziehbar ist. Dieser Forderung liegen mehrere Beobachtungen aus der Praxis zugrunde: zum einen die Feststellung, dass ein Großteil der Entwicklungszeit auf die Anpassung oder Änderung von bestehendem Quellcode entfällt, und zum anderen, dass funktionierender Code nicht automatisch lesbar, verständlich und leicht anpassbar sein muss. Besonders unter hohem Zeitdruck erstellte Programmteile fallen in diese Kategorie.

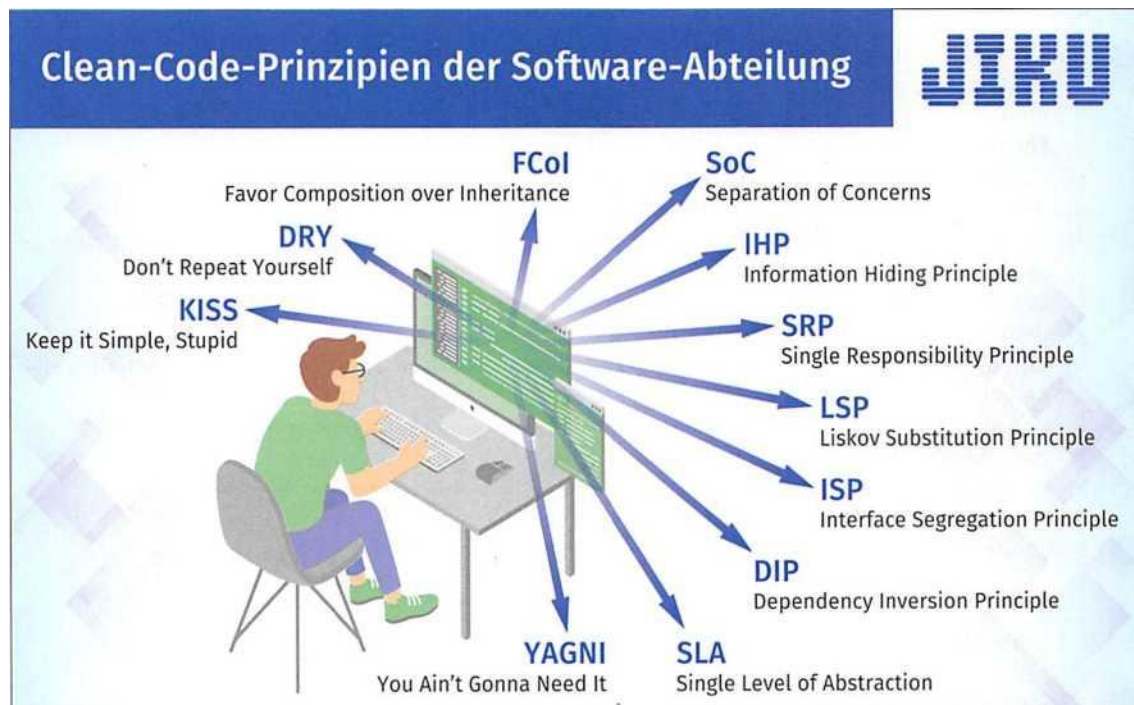


Den Quellcode nach der Fertigstellung einem Refactoring zu unterziehen, das wird zwar häufig eingeplant, aber in der Praxis selten umgesetzt, weil neue Projekte warten. Das führt bei Wartungs- und Erweiterungsaufgaben an diesem Programm schnell zu Fehlern und in vielen Fällen wird der Code noch unübersichtlicher und unverständlicher. Im besten Fall kostet es nur viel Zeit und Geld. Im schlechtesten Fall kann es dazu führen, dass die Software nicht mehr weiterentwickelt wird und veraltet.

Refactoring

Unter Refactoring wird eine Strukturverbesserung von Quellcode verstanden, wobei aber die Programmfunktionalität nicht verändert wird. Ziel ist es, die Lesbarkeit, Verständlichkeit, Wartbarkeit und Erweiterbarkeit des Quellcodes zu verbessern.

Ein entscheidendes Mittel, um solchen Fällen vorzubeugen, ist, gleich von Anfang an „sauberen“ Code zu schreiben. Um dieses zu erreichen, wurde eine Reihe von Prinzipien formuliert, welche bei der Entwicklung unterstützen sollen.



Diese Prinzipien sind zunächst einmal als Grundsätze zur Strukturierung von Software zu verstehen. Es sollte versucht werden, so viele dieser Prinzipien wie möglich zu beachten. Aber auch das Umsetzen von einem oder zwei Prinzipien verbessert meistens schon erheblich die Qualität des Quellcodes. Allerdings muss auch dazu gesagt werden, dass es sich dabei um Prinzipien handelt. Welcher Grad der Umsetzung zu einem „sauberen“ Code führt, ist in vielen Fällen rein subjektiv. Was von einem Entwickler bereits als einfach und unkompliziert angesehen wird, wird von einem anderem Entwickler als komplex eingestuft. Wenn man sich aber intensiver mit Clean Code beschäftigt, bekommt man mit der Zeit eine Grundidee davon, was von den meisten Programmierern als sinnvoll erachtet wird. Nachfolgend werden die Clean- Code-Prinzipien näher erläutert.

Clean-Code-Prinzipien

Keep It Simple and Stupid (KISS)

Das KISS-Prinzip besagt, dass der Code so einfach wie möglich sein sollte. Unnötige Komplexität sollte vermieden werden. Von zwei möglichen Lösungen sollte immer die einfachere gewählt werden.

Don't Repeat Yourself (DRY)

Das DRY-Prinzip zielt darauf ab, Redundanzen im Quellcode zu vermeiden oder zumindest zu reduzieren. Mehrfach verwendeter Code sollte in eigene Funktionen oder Klassen ausgelagert werden. Idealerweise kommt jede Funktionalität und Information nur einmal im System vor.

Favour Composition over Inheritance (FCol)

Das FCol-Prinzip besagt, dass Kompositionen anstelle von Vererbungen verwendet werden sollen. Dadurch sollen die Klassen entkoppelt werden, was zu einem flexibleren und stabileren Programm führt. Bevor Vererbung eingesetzt wird, sollte immer geprüft werden, ob nicht das gleiche Ergebnis mit einer Komposition erreicht werden kann.

Single Level of Abstraction (SLA)

Das SLA-Prinzip besagt, dass verschiedene Abstraktionslevel von Funktionen nicht miteinander vermischt werden sollen, um so die Lesbarkeit, das Verständnis und die Wartbarkeit des Codes zu erhöhen.

Single Responsibility Principle (SRP)

Das SRP besagt, dass jedes Modul (z. B. Klasse oder Funktion) genau eine Aufgabe erfüllen soll. Alles in diesem Modul soll sich nur auf diesen Zweck beziehen. Eine „universelle“ Funktion oder Klasse soll vermieden werden.

Separation of Concerns (SoC)

Beim SoC-Prinzip wird der Code in verschiedene Module unterteilt, die jeweils unterschiedliche Funktionen mit möglichst geringer Überlappung aufweisen.

Dependency Inversion Principle (DIP)

Das DIP gibt vor, dass Module höherer Ebenen nicht von Modulen niedriger Ebenen abhängen sollen. Die Abhängigkeit sollte allein auf Abstraktionen beziehungsweise auf Interfaces basieren.

Interface Segregation Principle (ISP)

Das ISP-Prinzip besagt, dass Interfaces so klein wie möglich sein sollten und auch nur das enthalten, was wirklich

benötigt wird. Kleinere Interfaces sind einfacher zu implementieren und erhöhen auch gleichzeitig die Flexibilität und die Wiederverwendungsmöglichkeit.

Open Closed Principle (OCP)

Das OCP besagt, dass eine Klasse offen für Erweiterungen, aber geschlossen gegenüber Modifikationen sein soll. Das heißt, die Funktionalität einer Klasse darf erweitert, aber nicht verändert werden.

Liskov Substitution Principle (LSP)

Das LSP besagt, dass sich Subtypen so verhalten müssen wie ihr Basistyp. Das heißt, ein Subtyp darf die Funktionalität des Basistyps erweitern, aber nicht einschränken. Oder anders ausgedrückt: Jeder Subtyp muss jederzeit auch als Basistyp verwendet werden können.

Information Hiding Principle (IHP)

Nach dem IHP soll das Innenleben von Modulen nach außen hin verborgen werden. Der Zugriff sollte über entsprechende Schnittstellen erfolgen.

Law of Demeter (LoD)

Nach dem LoD-Prinzip sollten Klassen in einem Projekt eine geringe Kopplung aufweisen, d. h., sie sollten nur mit wenigen Klassen aus ihrer Umgebung kommunizieren. Damit sollen komplexe Abhängigkeiten innerhalb des Programms vermieden werden.

Tell, Don't Ask Principle (TDA)

Beim TDA-Prinzip geht es darum, Daten und die dazugehörigen Funktionen in einem Modul zusammenzufassen. Anstatt von einer Klasse Daten anzufordern, um diese dann zu verarbeiten, sollte der Klasse besser mitgeteilt werden, was sie mit den Daten tun soll.

You Ain't Gonna Need It (YAGNI)

Das YAGNI-Prinzip besagt, dass nur das programmiert werden sollte, was wirklich gebraucht wird, und nicht zusätzlich noch das, was vielleicht mal gebraucht werden könnte. Dieses Prinzip ist eng verbunden mit der agilen Softwareentwicklung.

Einhaltung von Source-Code-Konventionen

Source-Code-Konventionen sind Leitlinien für die Erstellung des Programms. Sie umfassen beispielsweise Namensregeln, Kommentare, Deklarationen, Statements, White Spaces etc. Diese Konventionen sind vor allem in größeren Softwareprojekten unverzichtbar. Sie erhöhen die Lesbarkeit und sorgen für Konsistenz im Quellcode.

Vorsicht vor Optimierungen

Optimierungen verursachen in vielen Fällen einen hohen Aufwand. Vielfach sind diese aber weder notwendig noch nützlich. Solange sie nicht wirklich notwendig sind, sollte man darauf verzichten.

Implementation gibt den Entwurf wieder

Oftmals laufen Entwurf und Implementation in der Praxis auseinander. Es sollte immer versucht werden, dass die Implementation nicht völlig losgelöst vom Entwurf existiert, d.h. dass beispielsweise Module, die getrennt geplant wurden, nicht plötzlich in einem Modul zusammengefasst werden.

Es wäre sicherlich von Vorteil, wenn alle Prinzipien bei der Programmentwicklung beachtet und umgesetzt würden. Dieses ist in der Praxis nicht immer einfach. Deswegen werden die Prinzipien SRP, OCP, LSP, ISP und DIP als grundlegende Prinzipien empfohlen, um einen „Clean Code“ zu erreichen. Diese Prinzipien werden auch die SOLID-Prinzipien genannt.



SOLID

Die Clean-Code-Prinzipien in der Praxis dauerhaft umzusetzen, ist keine leichte Aufgabe, auch wenn diese leicht zu verstehen sind. Zum einen liegen nicht immer klare und präzise Anforderungen an das Softwareprodukt vor, was das Einhalten von einigen Prinzipien unmöglich macht. Zum anderen liegt es an mangelnder Disziplin oder schlicht an fehlender Erfahrung der Programmierer. Im letzteren Fall hilft es nur, sich kontinuierlich mit den Prinzipien vom Clean Code auseinanderzusetzen und saubere Implementierungen anzustreben. Je häufiger man die Prinzipien umsetzt, desto automatisierter werden diese.

Neben der Umsetzung der Clean-Code-Prinzipien hilft in der Praxis bereits die Beachtung einiger schnell umsetzbarer Tipps, um wartbareren Code zu schreiben. Diese betreffen vor allem Code-Konventionen und können auch in den kleinsten Programmen angewendet und trainiert werden.



Es gibt eine Clean-Code-Developer-Bewegung, die das Ziel verfolgt, durch ein einheitliches und überschaubares Regelwerk den Clean-Code-Ansatz interessierten Entwicklern nahezubringen.

Initiative für mehr Professionalität in der Softwareentwicklung

<https://clean-code-developer.de/>

✓ Kompetenzcheck

1. Geben Sie an, welche Aussagen richtig sind.
 - a. Die Idee hinter dem Clean-Code-Ansatz ist es, den Quellcode lesbarer, verständlicher und damit wartbarer zu gestalten.
 - b. Unter Refactoring kann allgemein eine Strukturverbesserung des Quellcodes verstanden werden.
 - c. Das KISS-Prinzip besagt, dass der Quellcode so einfach wie möglich gehalten werden soll.
 - d. Ein Prinzip des Clean Codes ist es, möglichst viel Funktionalität in ein Modul unterzubringen, um so einen universellen Einsatz zu gewährleisten.
 - e. Wo immer es geht, sollte die Funktionalität des Programms optimiert werden, um so dem Softwarequalitätskriterium „Effizienz“ gerecht zu werden.
 - f. Kompositionen sollten, wenn möglich, den Vererbungen vorgezogen werden.
 - g. SOLID umfasst die Clean-Code-Prinzipien ISP, OCP, DIP, LSP und SRP.
 - h. „Magie Numbers“ haben positive Auswirkungen auf die Wartbarkeit des Programms.
 - i. Es sollten „sprechende“ Namen für Bezeichner verwendet werden.
 - j. Jede Codezeile im Programm sollte nach Möglichkeit kommentiert werden.
2. Diskutieren Sie im Klassenverband die Praxistipps und Ihre Erfahrungen damit. Nennen Sie weitere Praxistipps aus Ihrer Programmiererfahrung.
3. Machen Sie sich Notizen, welche Programmierkonventionen (Styleguide) in Ihrer Firma eingehalten werden müssen. Stellen Sie diese in der Klasse vor.
4. Bearbeiten Sie im Handout die Aufgaben 8 und 9 der Lernsituation 1 und werten Sie u.a. einen englischen Text über die Namen von Bezeichnern aus.

2. Datenstrukturen von Softwarekomponenten planen und implementieren

- **Sie sollen aus einer gegebenen Anforderungsspezifikation die notwendigen Datenstrukturen ableiten und von einer Programmiersprache vorgegebene Datenstrukturen verwenden.**

Ein erster wichtiger Schritt bei der Planung und Implementierung von Softwarekomponenten ist die Bestimmung geeigneter Datenstrukturen, denn jedes Programm muss Daten verarbeiten und verwalten (siehe auch EVA-Prinzip). Die Auswahl der richtigen Datenstrukturen ist in vielen Fällen sogar wichtiger als die Implementierung der Algorithmen, weil diese in den meisten Fällen sehr einfach sind.



Grundsätzlich können die zu verarbeitenden Daten in verschiedenen Datenstrukturen verwaltet werden. Eine Datenstruktur als solches organisiert zunächst einmal nur Daten und stellt eventuell noch grundlegende Operationen, z.B. das Einfügen, Ändern oder Suchen von Daten, zur Verfügung. Die Aufgabe besteht nun darin, bezogen auf die Art der vorliegenden Daten (siehe auch Datenarten) und die zu lösende Aufgabe effiziente Datenstrukturen auszuwählen. Die Effizienzkriterien beziehen sich dabei zum einen auf den Speicherplatzverbrauch der Daten und andererseits auf die Geschwindigkeit, mit denen die erwähnten grundlegenden Operationen ausgeführt werden. Um die richtigen Datenstrukturen für seine Softwarekomponente auszuwählen, muss man die Konzepte und Funktionsweisen sowie die Vor- und Nachteile von verschiedenen Datenstrukturen kennen und diese sorgfältig gegeneinander abwägen.

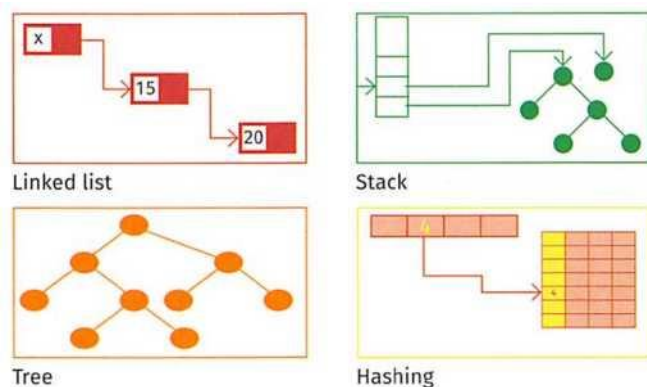
In diesem Abschnitt werden vor allem grundlegende Datenstrukturen betrachtet, welche für imperative Programmierparadigmen geeignet sind und dort zum Einsatz kommen. Dazu werden zunächst die Datenstrukturen im ersten Abschnitt dieses Kapitels programmiersprachenunabhängig miteinander verglichen und die Vor- und Nachteile näher beleuchtet. Allerdings müssen die hier besprochenen Datenstrukturen in der Regel nicht selbst implementiert werden, sondern man kann auf die mitgelieferten Lösungen der jeweiligen Programmiersprache zurückgreifen. Dieses wird danach an ausgewählten Beispielen in Python und Java vorgestellt.

Dennoch stellt die Implementierung von einfachen Datenstrukturen eine sehr gute Übung zum Erweitern der eigenen Programmierfähigkeiten dar. Zudem wird außerdem auch das Verständnis für deren Funktionsweise erhöht, weshalb solche Beispiele und Übungen in das Arbeitsbuch mit eingebaut wurden.

2.1. Grundlegende Datenstrukturen beschreiben

- **Sie sollen grundlegende Datenstrukturen unterscheiden und beschreiben können.**

Datenstrukturen spielen bei der Programmentwicklung eine große Rolle und ein Ziel sollte es sein, die Datenstrukturen für das Programm so auszuwählen, dass ein optimaler Zugriff und eine optimale Verwaltung der Daten gewährleistet sind. Bei dieser Entscheidungsfindung spielen u.a. die Struktur der Daten, die Datenmenge und die Anforderungen an die Datenverwaltung eine große Rolle. So sind nicht alle Datenstrukturen für alle Anforderungen gleich gut geeignet.

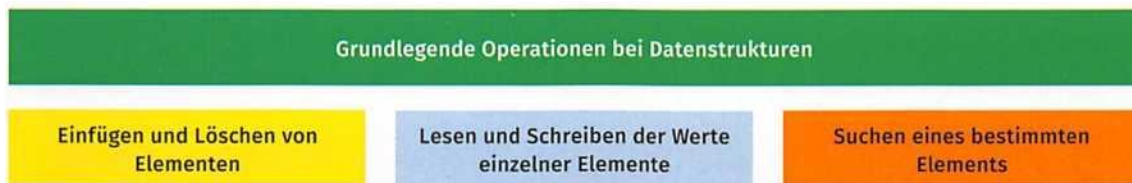


Datenstruktur

Unter einer Datenstruktur wird ein Objekt verstanden, in dem Daten mit gleichen oder unterschiedlichen Datentypen in einer bestimmten Art und Weise angeordnet und verwaltet werden.

Im Laufe der Zeit wurden verschiedene Datenstrukturen entwickelt, um verschiedenen Anforderungen gerecht zu werden. Grundlegende Kenntnisse über die verschiedenen Datenstrukturen und deren Vor- und Nachteile helfen bei der Programmentwicklung, die optimale Datenstruktur auszuwählen.

Man spricht von einer Datenstruktur, weil die Daten in diesem Objekt in einer bestimmten „Struktur“ miteinander verknüpft sind. Diese Struktur sollte man kennen. Datenstrukturen müssen verschiedene grundlegende Operationen (Funktionen) zur Verfügung stellen, mit deren Hilfe die Daten verwaltet werden.



Die Effizienz einer Datenstruktur ergibt sich im Allgemeinen aus dem Speicherbedarf und der Geschwindigkeit, mit der einzelne Operationen ausgeführt werden. Dabei können einzelne Operationen auch unterschiedlich effizient ablaufen. So sind bei einigen Datenstrukturen das Lesen und Schreiben von Werten sehr effizient, dafür sind aber das Einfügen und Löschen von Elementen sehr aufwendig. Andere Datenstrukturen benötigen mehr Speicherplatz für die Daten, weisen dafür aber beispielsweise erhebliche Geschwindigkeitsvorteile bei der Suche von bestimmten Elementen auf.

Arten von Datenstrukturen

Lineare Datenstrukturen

Bei einer linearen Datenstruktur werden die einzelnen Elemente sequenziell angeordnet. Dadurch können alle Elemente nacheinander durchlaufen werden. Beispiele für lineare Datenstrukturen sind das Array (Feld), der Stack (Stapel), die Queue (Warteschlange) oder die Linked List (verkettete Liste). Es wird zwischen statischen und dynamischen linearen Datenstrukturen unterschieden. Bei den statischen Varianten wird deren Größe einmal festgelegt und ändert sich nicht mehr während der Laufzeit des Programms. Als Hauptvertreter ist hier das Array zu nennen. Es gibt auch dynamische Arrays. Allerdings ist hier das Hinzufügen von neuen Elementen im Vergleich zu Listen teilweise sehr ineffizient.

Array	0	1	2	3	4	Index
	10	20	30	40	50	Werte

Zu den dynamischen linearen Datenstrukturen zählen der Stapel, die Warteschlange und die verkettete Liste. Hier kann sich die Größe der Datenstruktur während der Laufzeit verändern und muss vorher nicht bekannt sein.



Im Vergleich zu den Arrays ist aber der Zugriff auf einzelne Elemente langsamer, da bis zu der Position jeweils immer die gesamte Liste durchlaufen werden muss.

Das Suchen nach bestimmten Elementen ist bei beiden Varianten gleich groß. Bei größeren Datenmengen ist die Suche nach einem Element, dessen Position nicht bekannt ist, oft zeitaufwendig. Die Suche wird dabei sequenziell Element für Element durchgeführt, bis das betreffende Element gefunden wurde. Hierfür eignen sich andere Datenstrukturen besser.

Hashbasierte Datenstrukturen

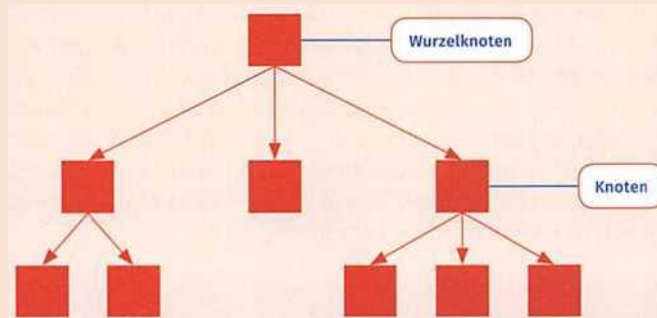
Bei hashbasierten Datenstrukturen wird eine Hashfunktion verwendet, um die Daten in einem Schlüssel-Wert-Paar abzubilden. Der Unterschied zu Arrays besteht darin, dass auf die einzelnen Elemente anstatt über Indizes über einen Schlüssel zugegriffen wird. Dabei wird der Indexwert eines Elementes über eine Hashfunktion generiert. Beispiele für hashbasierte Datenstrukturen sind die Hashtable oder der HashTree. Diese Datenstruktur ist nicht linear.

Schlüssel	Wert
Deutschland	Berlin
Frankreich	Paris
Spanien	Madrid

Hashbasierte Datenstrukturen bieten einen effizienten Zugriff auf die Datenwerte über den Schlüssel (Lesen und Schreiben). Dadurch ist ein schnelles Suchen auch in großen Datenmengen möglich. Ein großer Nachteil ist, dass verschiedene Schlüssel durchaus zu gleichen Indizes führen können, was sogenannte Kollisionen erzeugt. Dieses stellt bei hashbasierten Datenstrukturen ein nicht unerhebliches Problem dar.

Bäume

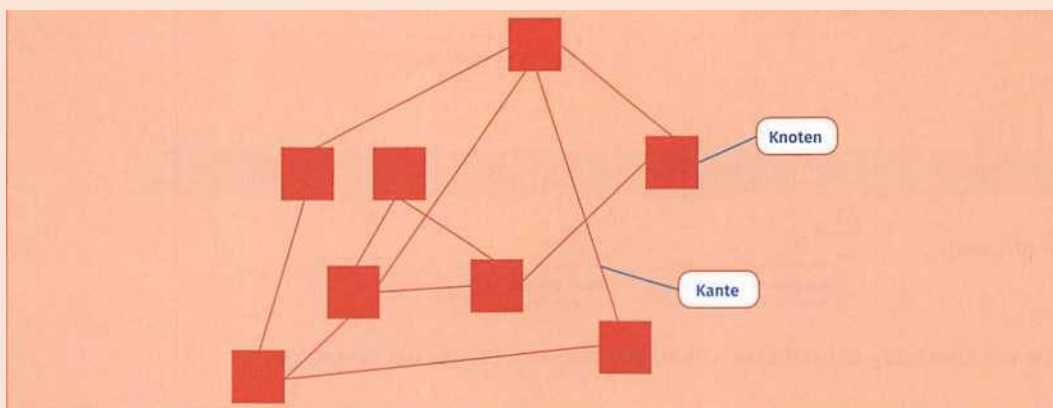
Eine Baumdatenstruktur ist eine hierarchische Daten Struktur. Es hat Knoten, die durch Kanten verbunden sind. Dabei können ausgehend von einem Wurzelement mehrere gleichartige Elemente miteinander verbunden werden. Bäume gehören zu den meistverwendeten Datenstrukturen in der Informatik, weshalb es viele Arten von Bäumen gibt. Beispiele für Bäume sind der Binärbaum, der binäre Suchbaum oder der Heap. Bei Binärbäumen zweigen maximal zwei Äste von einem Knoten ab und in Suchbäumen sind die Elemente in der Baumstruktur geordnet abgelegt, sodass man schnell Elemente finden kann. Bäume werden u.a. in der Bildverarbeitung, Sprachverarbeitung und künstlichen Intelligenz verwendet.



Graphen

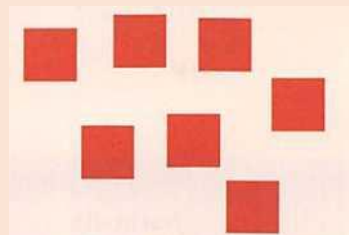
Ein Graph ermöglicht es, Elemente zu verwalten, die miteinander in einer Beziehung stehen, ohne dass die Struktur linear oder baumartig ist. Ein Graph ist eine nicht lineare, bildliche Darstellung von Daten, die aus Kanten und Knoten bestehen. Kanten sind die Verbindungen zwischen den einzelnen Elementen (Knoten). Beispiele für Graphen in der Informatik sind die Adjazenzmatrix, die Inzidenzmatrix oder Adjazenzliste.

Es gibt viele interessante und praxisrelevante Problemstellungen, die typischerweise mithilfe von Graphen beschrieben und gelöst werden können (siehe auch Graphendatenbanken).



Mengen

Eine Menge stellt eine ungeordnete Ansammlung von Elementen dar, die nicht miteinander in Beziehung stehen. Dabei sind die Elemente vom gleichen Datentyp und es darf nur jeweils ein Exemplar eines Elements in einer Menge vorhanden sein. Auf Mengen können in der Regel Operationen wie Vereinigung, Differenz oder Schnittmengenbildung angewandt werden. Wenn Programmiersprachen Mengen als Datenstruktur integriert haben, dann werden diese typischerweise als „Set“ bezeichnet.



Diese Übersicht beinhaltet nur die grundlegendsten Datenstrukturen. Es gibt noch weitere Datenstrukturen, welche meistens aber für spezielle Anwendungsfälle konzipiert sind. Nachfolgend werden einige typische Vertreter von linearen und hashbasierten Datenstrukturen genauer betrachtet, weil diese ein großes Anwendungsgebiet besitzen und in der einen oder anderen Form in allen Programmiersprachen integriert wurden.

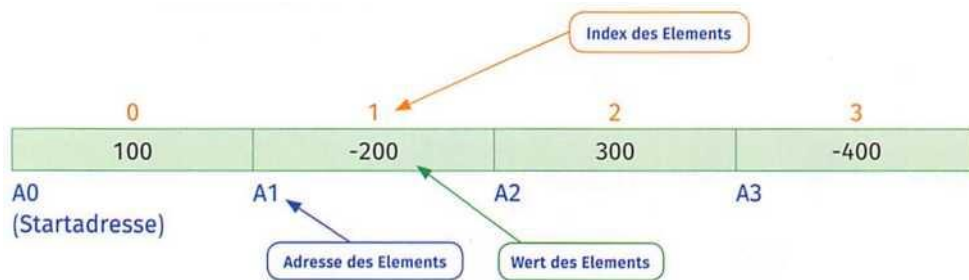
(1) Felder (Array)

Die einfachste Datenstruktur ist ein Array. Es werden hierbei mehrere Elemente vom selben Datentyp gespeichert. Der Zugriff auf die einzelnen Elemente wird über einen Index gewährleistet. Dabei wird der Index nicht mitgespeichert und dient nur zur Ermittlung der Speicheradressen der Elemente.

Technisch gesehen wird der angegebene Index mit der Größe des jeweiligen Datentyps multipliziert und zu der Startadresse des Arrays im Speicher addiert, um die Adresse des gesuchten Elements zu erhalten. Die Größe eines Arrays kann zur Laufzeit des Programms nicht ohne größeren Aufwand geändert werden, d. h., die Anzahl der zu speichernden Elemente sollte bekannt sein und nicht zu groß schwanken. Ein Array, welches beispielsweise für 1 000 Elemente ausgelegt ist, aber den größten Teil der Zeit nur 30 Elemente enthält, kann eine Speicherverschwendung darstellen. Für solche Anforderungen sind andere Datenstrukturen besser geeignet.

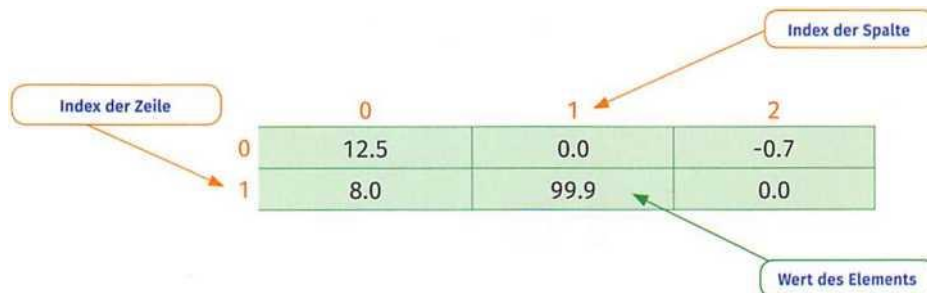
Ein Array kann eindimensional oder mehrdimensional sein. In der Praxis werden vor allem eindimensionale Arrays (Vektoren) und zweidimensionale Arrays (Tabelle oder Matrix) verwendet. Wegen ihrer Einfachheit und grundlegenden Bedeutung ist diese Datenstruktur bei den allermeisten Programmiersprachen im Sprachumfang integriert.

Eindimensionales Array (Vektor)



Adresse des Elements = Startadresse + (Index des Elements * Größe des Datentyps)

Zweidimensionales Array (Matrix)



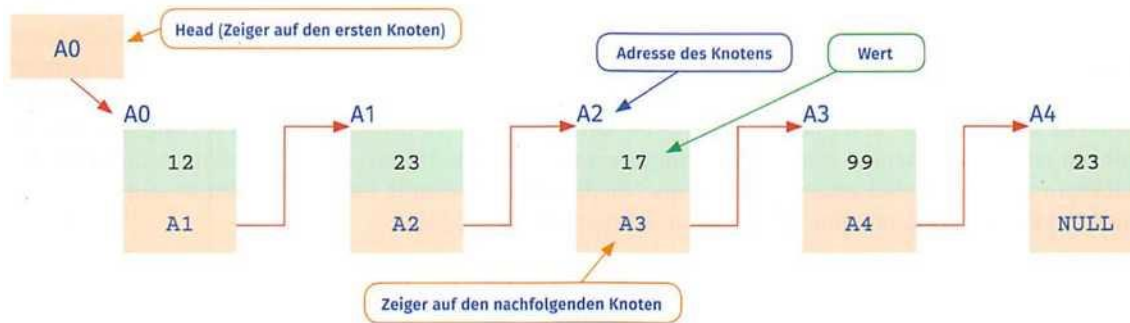
Vor- und Nachteile von Arrays	
Vorteile	Nachteile
• schnelles Lesen und Schreiben der Werte von Elementen, da direkt über die Indizes darauf zugegriffen werden kann • einfache Datenstruktur • ist in allen gängigen Programmiersprachen standardmäßig integriert	• Größe muss im Vorfeld festgelegt werden • in der Regel kein Hinzufügen von neuen Elementen möglich • bei dynamischen Arrays entsteht teilweise ein sehr hoher Aufwand beim Hinzufügen von neuen Elementen; auf das Löschen wird in der Regel ganz verzichtet • teilweise aufwendiges Suchen notwendig, weil im ungünstigsten Fall alle Elemente durchlaufen werden müssen

(2) Einfach verkettete Listen (Linked List) und doppelt verkettete Listen (Double Linked List)

Verkettete Listen sind lineare Datenstrukturen, welche zur dynamischen Speicherung von beliebig vielen Elementen dienen. Demnach kann die Größe von Listen während der Laufzeit wachsen bzw. schrumpfen. Dabei erhält jedes Element (Knoten) neben dem eigentlichen Wert noch einen Verweis auf die Adresse des nächsten Elements. Somit entsteht eine Kette von Elementen. Eine Liste verbraucht mehr Speicherplatz als ein Array für ein Element. Auch der Zugriff auf Elemente zum Lesen und Schreiben ist im Durchschnitt langsamer als beim Array, da bis zu dem entsprechenden Element immer die gesamte Liste durchlaufen werden muss. In der Regel werden einfach verkettete und

doppelt verkettete Listen unterschieden. Bei einer einfach verketteten Liste enthält jeder Knoten einen Zeiger auf den nachfolgenden Knoten. Der letzte Knoten einer Liste zeigt entweder auf sich selbst oder auf NULL.

Aufbau einer einfach verketteten Liste

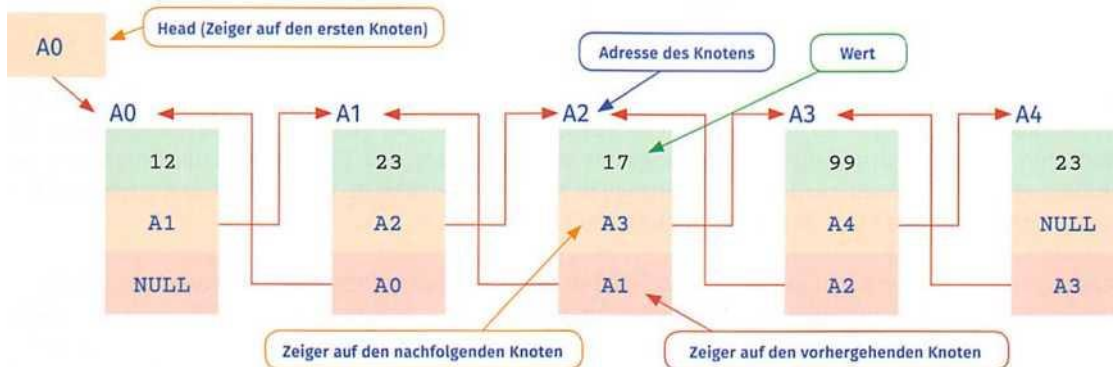


Vor- und Nachteile von einfach verketteten Listen

Vorteile	Nachteile
• Elemente können sehr schnell am Anfang der Liste eingefügt werden • Einfügen und Löschen von Elementen sind schnell; allerdings muss die entsprechende Stelle in der Liste zunächst gesucht werden • geringer Speicherplatzbedarf • Größe muss im Vorfeld nicht bekannt sein	• Lesen und Schreiben von Werten sowie das Suchen können bei großen Datenmengen zeitintensiv sein; Zeitaufwand hängt davon ab, an welcher Stelle sich das Element in der Liste befindet

Doppelt verkettete Listen sind eine Erweiterung der einfach verketteten Listen. Bei diesen erhält jeder Knoten noch zusätzlich einen Zeiger auf seinen Vorgänger. Da der erste Knoten keinen Vorgänger hat, zeigt dieser entweder auf sich selbst oder wieder auf null.

Aufbau einer doppelt verketteten Liste

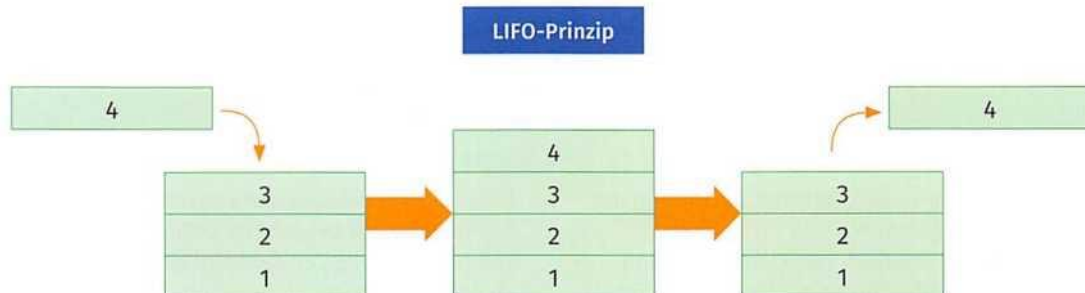


Vor- und Nachteile von doppelt verketteten Listen

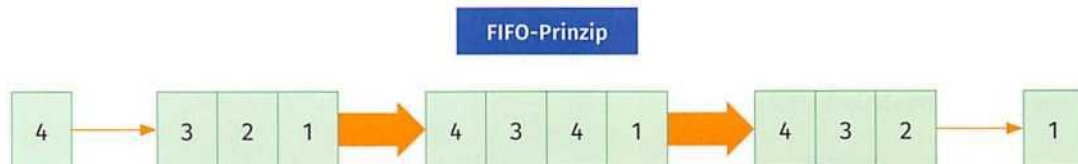
Vorteile	Nachteile
• Einfügen und Löschen von Elementen sind schnell • einzelne Elemente sind schneller zu finden • geringerer Speicherplatzbedarf als Arrays, da die Größe der Liste immer der aktuellen Anzahl der gespeicherten Werte entspricht • Liste kann bei Bedarf von beiden Seiten durchlaufen werden • Größe muss im Vorfeld nicht bekannt sein	• höherer Speicherplatzbedarf gegenüber einer einfach verketteten Liste wegen dem zusätzlichen Zeiger • beim Einfügen und Löschen müssen sowohl Vorgänger als auch Nachfolger angepasst werden

(3) Stapel (Stack) und Warteschlange (Queue)

Der Stack ist eine Datenstruktur, welcher wie ein Stapel funktioniert. Neue Elemente werden oben auf dem Stapel abgelegt und es kann immer nur auf das oberste Element vom Stapel zugegriffen werden (Lesen oder Entfernen). Dieses Prinzip ist auch als LIFO (Last In First Out) bekannt. Der Stack ist eine sehr einfache Datenstruktur und findet vielfältige Verwendung in der Informatik. Eine der wichtigsten Anwendungen ist der sogenannte Aufruf-Stack. Auf diesem werden beispielsweise bei Funktionsaufrufen die Rücksprungadressen und lokale Variablen der Funktionen gespeichert. Rekursive Funktionsaufrufe wären ohne einen Aufruf-Stack gar nicht möglich.



Das Gegenstück zum Stack ist die Queue (Warteschlange). Im Gegensatz zum Stack findet hier das Prinzip FIFO (First In First Out) Anwendung. Auf jenes Element, welches als Erstes in der Warteschlange gespeichert wurde, kann auch als Erstes wieder zugegriffen werden. Neue Elemente werden am Ende der Warteschlange angefügt.



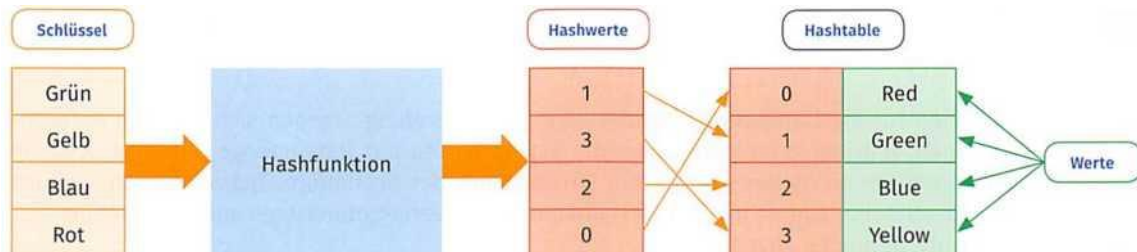
Anwendung finden Warteschlangen beispielsweise bei grafischen Benutzeroberflächen, welche Tastatur- und Mausevents in einer Message-Queue speichern und in der Folge ihres Auftretens abarbeiten. Weiterhin werden auch auf diese Weise langsame externe Geräte, z.B. Drucker, von der Programmabarbeitung entkoppelt. Dazu werden Druckaufträge in einer Warteschlange gespeichert und nach und nach abgearbeitet, wenn der Drucker zur Verfügung steht.

Stacks und Queue werden in der Regel durch Listen realisiert und stellen spezielle Varianten von diesen dar. Deswegen treffen zunächst einmal alle Vor- und Nachteile zu, welche auch für Listen gelten. Da allerdings nur auf das letzte bzw. das erste Element der Datenstruktur zugegriffen wird und auch nicht gesucht wird, relativieren sich die Nachteile sowie die Vorteile wie z.B. das effiziente Einfügen von Elementen innerhalb der Datenstruktur.

(4) Hashtabelle (Hashtable)

Die Hashtabelle oder auch Streuwerttabelle ist eine spezielle Indexstruktur, bei welcher die Speicherposition der einzelnen Elemente direkt durch eine Hashfunktion berechnet wird. Die Elemente einer Hashtabelle stellen immer ein Schlüssel-Wert-Paar dar. Über den Schlüssel kann direkt auf die gespeicherten Werte zugegriffen werden. Hashfunktionen sind jedoch nicht eindeutig, sodass zwei verschiedene Schlüssel denselben Hashwert erzeugen können. Diesen Fall nennt man Kollision. Eine Kollision benötigt immer eine gesonderte Behandlung und kann die Zugriffszeiten erheblich ausbremsen.

Im Beispiel soll über das deutsche Wort als Schlüssel auf den englischen Begriff zugegriffen werden.



Vor- und Nachteile von Hashtabellen	
Vorteile	Nachteile
• schnelles Suchen in großen Datenmengen möglich • schneller Lese- und Schreibzugriff, vor allem im Vergleich mit Bäumen • weniger Speicherplatzverbrauch im Vergleich zu Arrays	• mögliche Kollisionen bei der Erzeugung von Hashwerten • mit steigendem Füllgrad steigt die Wahrscheinlichkeit einer Kollision

✓ Kompetenzcheck

- Geben Sie an, welche Aussagen richtig sind.
 - Eine Datenstruktur ist ein Objekt, in dem Daten auf eine bestimmte Art und Weise geordnet und verwaltet werden.
 - Bäume zählen zu den linearen Datenstrukturen.
 - Klassische Arrays können während der Laufzeit nicht ihre Größe ändern.
 - Bei verketteten Listen sind Einfüge- und Löschoperationen sehr langsam.
 - Doppeltverkettete Listen ermöglichen eine schnelle Suche von Elementen.
 - Bäume zählen zu den dynamischen Datenstrukturen, d.h. sie können zur Laufzeit wachsen und schrumpfen.
 - Auf „Sets“ können Mengenoperationen angewandt werden.
 - Es kann nur ein- oder zweidimensionale Arrays geben.
 - Einfach verkettete Listen besitzen einen Zeiger auf ihren Vorgänger.
 - Hashtabellen ermöglichen eine schnelle Suche von Elementen in großen Datenmengen.
 - Bei der Berechnung des Hashwertes kann es zu Kollisionen kommen.
 - Warteschlangen (Queue) funktionieren nach dem LIFO-Prinzip.
 - Bei Stacks kann nur auf das zuletzt hinzugefügte Element zugegriffen werden.
- Diskutieren Sie im Klassenverband die Vor- und Nachteile von einzelnen Datenstrukturen und berichten Sie über die Erfahrungen, welche Sie damit gesammelt haben.
- Bearbeiten Sie im Handout die Aufgabe 1 der Lernsituation 2 und implementieren Sie einfache Datenstrukturen wie eine einfach verkettete Liste.

2.2. Datenstrukturen bedarfsgerecht auswählen

- **Sie sollen geeignete Datenstrukturen entsprechend einer Anforderungsspezifikation auswählen und Ihre Entscheidung fachlich begründen.**

Die Auswahlkriterien für die Datenstrukturen der eigenen Anwendung ergeben sich aus den Anforderungsspezifikationen. Hierbei spielen vor allem die Eigenschaften der Datenmenge eine Rolle, z.B. die Anzahl der Daten und deren Dynamik sowie die Effizienz, mit der bestimmte Operationen ausgeführt werden sollen, z.B. schneller Zugriff. In der Regel wird mit Arrays, verketteten Listen und Hashtabellen der größte Teil aller Anforderungen abgedeckt.

Nachfolgend sind einige Fallbeispiele aufgeführt, bei denen aufgrund der Anforderungen bestimmte Datenstrukturen bevorzugt werden.

(1) Fallbeispiel 1

Softwareprojekt „Yachthafen Resort“ Sitzplatzbelegung

Die Software soll ein Modul für das Eventmanagement des Theaters enthalten, das zum „Yachthafen Resort“ gehört. In diesem Rahmen soll eine Komponente entwickelt werden, mit deren Hilfe die jeweils aktuelle Sitzplatzbelegung für das Theater erfasst werden kann.

Der Zuschauerraum des Theaters hat eine feste Anzahl an Sitzreihen mit einer festen Anzahl an Sitzplätzen pro Reihe, welche durch eine Reihennummer und eine Sitzplatznummer in der Reihe identifiziert werden, z. B. Reihe 3, Sitzplatz 12.





Bei der Analyse der Ausgangssituation ergeben sich folgenden Fakten:

1. Es sollte sich um eine zweidimensionale Datenstruktur handeln, da so leichter die einzelnen Sitze über die Reihe und die Sitzplatznummer angesprochen werden können.
2. Eine sich dynamische ändernde Anzahl an Daten liegt nicht vor.
3. Die Anzahl der Daten steht fest bzw. ist dem Benutzer des Programms bekannt und kann eingegeben werden.
4. Es sollen vor allem das Lesen und das Schreiben von Daten effizient ausgeführt werden.
5. Ein effizientes Suchen nach Daten ist nicht notwendig.

Auf Grundlage dieser Analyse kann nun eine entsprechende Datenstruktur ausgewählt werden, welche den Anforderungen am meisten entspricht. Da die Anzahl der Daten bekannt ist und sich nicht dynamisch verändert, bietet sich ein zweidimensionales Array an, welches eine einfache Datenstruktur ist und effizientes Lesen und Schreiben zulässt.

(2) Fallbeispiel 2

Bei der Analyse der Ausgangssituation ergeben sich folgenden Fakten:

Softwareprojekt „Yachthafen Resort“ Getränkebestellung

Die Software soll ein Modul für die Bestellungen im Restaurant enthalten, das zum „Yachthafen Resort“ gehört.

In diesem Rahmen soll eine Komponente entwickelt werden, mit deren Hilfe Getränkebestellungen im Restaurant erfasst und der Reihe nach abgearbeitet werden.

Dabei erfasst das Personal die gewünschten Getränke (Art, Anzahl) und den Tisch in einer mobilen App. Die Daten werden zum Tresen übermittelt, an dem mehrere Barkeeper die Getränke der Reihe nach zubereiten und dazu als bearbeitet markieren. Dadurch wird die Bestellung aus der Bestellliste entfernt.





1. Die Anzahl der zu verwaltenden Daten ist nicht bekannt.
2. Die Anzahl der Daten ändert sich dynamisch.
3. Die Daten werden in der Reihenfolge nach dem Eingang abgearbeitet (FIFO-Prinzip).
4. Vor allem das Einfügen und das Löschen von Daten müssen effizient sein.
5. In seltenen Fällen werden die Daten einer Bestellung nochmal geändert.

Auf Grundlage dieser Analyse kann nun eine entsprechende Datenstruktur ausgewählt werden, welche den Anforderungen am meisten entspricht. Da die Anzahl der Daten nicht bekannt ist und sich dynamisch verändert, bietet sich

eine Queue an, welche nach dem FIFO-Prinzip arbeitet und effizientes Einfügen und Löschen von Daten gewährleistet. Aber auch eine doppelt verkettete Liste ist möglich.

(3) Fallbeispiel 3

Softwareprojekt „Yachthafen Resort“

Grafikmodul

Die Software soll um ein Grafikmodul erweitert werden, welches an verschiedenen Stellen in den anderen Programmteilen zum Einsatz kommt.

Als Teilaspekt dieser Aufgabe sollen in dem Modul Informationen über RAL-Farben (standardisierte Farben) gespeichert werden. Dabei sind folgende Angaben pro Farbe interessant:

RAL-Farbnummer der Farbe (Integerzahl),
deutsche Bezeichnung, englische Bezeichnung,
spanische Bezeichnung, RGB-Farbcode

Beispiel: 1005, Honiggelb, Honey yellow, Amarillo miel, #c89f04

Die Daten der einzelnen Farben ändern sich kaum, aber es soll sehr schnell nach den Daten einzelne Farben gesucht und darauf zugegriffen werden können. Dieses soll über die RAL-Farbnummer geschehen.




Bei der Analyse der Ausgangssituation ergeben sich folgende Fakten:

1. Die Anzahl der Daten ist bekannt, aber diese werden erst mit der Zeit eingegeben, sodass die Datenstruktur mit der Zeit wächst.
2. Es handelt sich um eine größere Anzahl von Daten.
3. Die Daten ändern sich nach der Eingabe sehr selten.
4. Es sollen eine effiziente Suche und ein schnelles Lesen der Daten möglich sein.
5. Der Zugriff erfolgt immer über die standardisierte RAL-Farbnummer.

Auf Grundlage dieser Analyse kann nun eine entsprechende Datenstruktur ausgewählt werden, welche den Anforderungen am meisten entspricht. Da es sich um eine größere Anzahl von Daten handelt und ein effizienter Zugriff auf die Daten über eine festgelegte RAL-Farbnummer erfolgen soll, bietet sich eine Hashtabelle als Datenstruktur an. Eine Hashtabelle bietet auch die Möglichkeit, sich dynamisch ändernde Datenmengen zu verwalten. Die erforderlichen Daten werden in einem Farbbjekt gespeichert und das Objekt dann in der Hashtabelle.

✓ Kompetenzcheck

1. Überlegen Sie sich in Partnerarbeit, welche Datenstruktur Sie für die nachfolgenden Situationen auswählen würden und begründen Sie Ihre Auswahl. Diskutieren Sie im Anschluss Ihre Ergebnisse im Klassenverband.
 - a. Ein Liegeplatz im Yachthafen Resort besitzt immer einen Status, z. B. vermietet, belegt oder gesperrt. Der aktuelle Status von jedem Liegeplatz soll in einer Datenstruktur verwaltet werden.
 - b. Es soll für geologische Messstationen ein Programm entwickelt werden, welches spezielle geologische Ereignisse protokolliert. Diese geologischen Ereignisse treten unregelmäßig auf und die Anzahl pro Tag kann nicht vorhergesagt werden. Es sollen alle Ereignisse eines Tages zunächst in einer Datenstruktur gespeichert werden. Diese werden einmal pro Tag dann automatisch durch das Programm nach gewissen Kriterien gefiltert und aufbereitet. Die aufbereiteten Daten werden dann an eine Datenbank übermittelt und alle Daten des Tages gelöscht.
 - c. Beim Start eines zu entwickelnden Programms sollen u.a. die internen Telefonnummern der einzelnen Mitarbeiter in eine Datenstruktur geladen werden. Die Telefonnummer soll schnell durch die Eingabe des Namens des Mitarbeiters gefunden werden.
2. Bestimmen Sie für weitere Situationen die entsprechende Datenstruktur. Bearbeiten Sie dazu in der Lernsituation 2 die Aufgabe 2.

2.3. Grundlegende Datenstrukturen von Python unterscheiden und einsetzen

➤ Sie sollen grundlegende Datenstrukturen in Python kennen und anwenden können.

Python stellt einige leistungsstarke Datenstrukturen zur Verfügung. Darunter fallen Listen, Dictionary, Tupel und Sets. Mit diesen in Python implementierten Datenstrukturen kann ein großer Teil der Anforderungen an moderne Programme abgedeckt werden. Diese Datenstrukturen sind als „Built-in“ schon in Python integriert und müssen nicht durch Module eingebunden werden.



Grundlegende Datenstrukturen in Python	
Liste	Die vielseitigste Datenstruktur in Python ist die Liste, welche hier eine Sammlung von verschiedenen Objekten darstellt. Dabei müssen die einzelnen Elemente einer Liste nicht vom gleichen Datentyp sein. Listen können während der Laufzeit des Programms beliebig wachsen und schrumpfen. Mithilfe von Listen lassen sich u.a. Arrays abbilden, wie man sie aus Java oder C++ kennt. Allerdings sind die Listen in Python diesbezüglich viel flexibler und leistungsfähiger.
Dictionary	Eine weitere wichtige Datenstruktur in Python ist das Dictionary. In einem Dictionary werden Elemente in Form von Schlüssel-Wert-Paaren abgespeichert. Auf ein einzelnes Element im Dictionary kann über einen eindeutigen Schlüssel zugegriffen werden. Wie eine Liste kann Dictionary beliebig wachsen und schrumpfen während der Laufzeit des Programms. Das Dictionary in Python ähnelt Hash oder Maps in anderen Programmiersprachen.
Tupel	Ein Tupel in Python ist eine unveränderliche, geordnete Sequenz von Elementen. Damit entspricht ein Tupel einer Python-Liste, mit dem Unterschied, dass ein Tupel nicht verändert werden kann. Um ein Tupel zu vergrößern oder zu verkleinern, muss ein neues Tupel erstellt werden.
Set	Ein Set stellt in Python eine ungeordnete Sammlung von einmaligen Elementen dar. Im Unterschied zum Tupel oder zur Liste kann in einem Set ein Element nicht mehrmals vorkommen. Auf ein Set können Standardoperationen für Mengen wie beispielsweise Vereinigung, Schnittmenge und Differenz angewandt werden.

Nachfolgend werden die Liste und das Dictionary genauer betrachtet und an Beispielen erläutert. Eindimensionale Listen in Python sind schon aus dem ersten Ausbildungsjahr bekannt. Deswegen werden an dieser Stelle nur die wichtigsten Informationen nochmals überblicksartig dargestellt. Etwas näher wird dann auf die mehrdimensionalen Listen eingegangen.

(1) Eindimensionale Liste

In einer eindimensionalen Liste in Python werden mehrere Werte hintereinander abgespeichert. Dabei müssen diese Werte nicht vom gleichen Datentyp sein. In einer Liste können sowohl Ganzzahlen, Kommazahlen als auch Strings gemeinsam gespeichert werden.

Grundlegende Operationen

Anlegen einer Liste

Angelegt wird eine Liste, indem eine Aufzählung ihrer Elemente in eckigen Klammern [] erfolgt.

Beispiele für das Anlegen von eindimensionalen Listen

```
</> temperaturen = [21.3, 2.0, 33.01, -23.22, 14.23]
gegenstände = ["Löffel", "Messer", "Gabel"]
hauptstädte = ["Berlin", 3644826, 891.68, "Paris", 2175601, 105.34]
```

Lesen und Schreiben von Werten

Auf die einzelnen Listenelemente kann über einen Index zugegriffen werden, welcher bei 0 beginnt.

Beispiele für den Zugriff auf Listenelemente

```
</> temperaturen[0] = 20.03

for monat in monate:
    print(monat)

print("Name: " + hauptstädte[0] + ", Einwohnerzahl: " +
      str(hauptstädte[1]))
```

Löschen eines Listenelementes

Das Löschen eines Elementes aus der Liste kann mit der Funktion *del* erfolgen. Hinter *del* wird das Element angegeben, welches gelöscht werden soll.

Beispiel für das Löschen eines Elementes

```
</> liste = [1, 2, 3]
print(liste)      # Ausgabe: [1, 2, 3]
del liste[1]      # Löschen des zweiten Listenelements
print(liste)      # Ausgabe: [1, 3]
```

Für die Arbeit mit Listen stehen noch weitere verschiedene Funktionen zur Verfügung.

Funktionen für die Arbeit mit Listen	
<code>append(value)</code>	hängt den Wert „value“ an das Ende der Liste an
<code>extend(liste)</code>	hängt alle Werte der Liste „liste“ an das Ende der bestehenden Liste an
<code>insert(index, value)</code>	fügt den Wert „value“ an der Stelle „index“ in die Liste ein
<code>pop(index)</code>	gibt den Wert des Elementes an der Stelle „index“ zurück und entfernt anschließend das Element; wird kein Index angegeben, dann wird das letzte Element entfernt
<code>remove(value)</code>	entfernt das erste Vorkommen des Wertes „value“ aus der Liste
<code>reverse()</code>	Kehrt die Reihenfolge der Elemente in der Liste um
<code>sort()</code>	sortiert die Liste

(2) Mehrdimensionale Liste

Neben eindimensionalen Listen ist es auch möglich, mehrdimensionale Listen anzulegen. Dabei ist die Anzahl der Dimensionen nicht begrenzt. Hier wird anhand einer zweidimensionalen Liste exemplarisch die Erzeugung und Verwendung von mehrdimensionalen Listen erläutert.

Eine zweidimensionale Liste kann man sich in Python so vorstellen, dass ein Listenelement selbst wieder eine Liste definiert. Die Elemente der Liste müssen auch hier nicht vom gleichen Typ sein. Man kann an eine Matrix oder eine Tabelle denken, um sich eine zweidimensionale Liste anschaulich vorzustellen. Jede Zelle einer zweidimensionalen Liste ist eindeutig durch die Position in einer Zeile und einer Spalte identifizierbar. Als Beispiel sollen die Werte folgender Matrix in einer zweidimensionalen Liste gespeichert und bearbeitet werden.

Zweidimensionale Beispielmatrix mit Werten

Index Zeile		0	1	2	Index Spalte
	0	13.5	-2.0	1.3	
	1	0.0	91.8	0.0	

Grundlegende Operationen

Anlegen von mehrdimensionalen Listen

Das Anlegen einer mehrdimensionalen Liste, in diesem Fall einer zweidimensionalen, entspricht dem Vorgehen bei einer eindimensionalen Liste, mit dem Unterschied, dass in die ersten eckigen Klammern noch weitere für die zweite Dimension gesetzt werden.

Anlegen der Beispielliste

```
</> spalten = 3
zeilen = 2
liste = [[0] * spalten] * zeilen
liste = [[13.5, -2.0, 1.3], [0.0, 91.8, 0.0]]
```

Lesen und Schreiben von Werten

Der Zugriff auf die einzelnen Elemente erfolgt wieder über die Angabe der entsprechenden Indizes. Bezogen auf eine zweidimensionale Liste müssen der Index der Zeile und der Index der Spalte angegeben werden. Der Index der Zeilen bzw. der Spalten beginnt immer mit dem Wert 0. Der letzte Index ist immer Anzahl der Zeilen bzw. Anzahl der Spalten minus 1. Wird ein Index verwendet, dessen Wert außerhalb des Listenbereichs liegt, wird ein Laufzeitfehler erzeugt.

Beispiele für den Zugriff auf die Listenelemente

```
</> print(liste[1][2]) # Ausgabe: 0.0

liste[0][2] = 45
print(liste[0][2]) # Ausgabe: 45

x = liste[0][0] * liste[1][1]
print(x) # Ausgabe: 1239.3
```

Einfügen und Löschen von Elementen

Funktionen, welche bei der Arbeit mit eindimensionalen Listen Anwendung finden, können auch für mehrdimensionale Listen verwendet werden. So kann für das Anhängen eines Elementes der append-Befehl und für das Löschen eines Elementes die del-Funktion verwendet werden. Dabei beziehen sich die einzelnen Funktionen immer nur auf die angegebene Zeile. Somit kann auch erreicht werden, dass mehrdimensionale Listen pro Zeile eine unterschiedliche Anzahl an Elementen aufweisen. Durch das Anhängen von einem Element in der Zeile 0 in dem Beispiel und durch das Löschen des zweiten Listenelements in der Zeile 1 würde nachfolgende Struktur erreicht werden.

Beispiel für das Anfügen und das Löschen von Elementen in mehrdimensionalen Listen

```
</> liste[0].append(11.11) # einen Wert in der Zeile 0 hinzufügen
del(liste[1][1]) # den zweiten Wert in der Zeile 1 löschen
print(liste[0][3]) # die Zeile 1 hat nun eine Spalte mehr. Index 3
# möglich
print(liste[1][1]) # die zweite Zeile hat nun eine Spalte weniger
print(liste) # alle Elemente der Liste anzeigen
# [[13.5, -2.0, 45, 11.11], [0.0, 0.0]]
print(liste[1][2]) # nun IndexError: list index out of range
```

	0	1	2	3
0	13.5	-2.0	1.3	11.11
1	0.0	0.0		

91.8 wurde gelöscht

neuer Wert

Allerdings bergen solche Strukturen auch die Gefahr, dass Indizes benutzt werden, welche keine Entsprechung in der Liste haben und somit eine Exception erzeugen.

(3) Dictionary

Ein Dictionary kann man wörtlich mit einem Wörterbuch vergleichen. Dabei ist einem Begriff ein anderer zugeordnet. Diese Zuordnung zweier Begriffe nennt man Paar. Ein Paar besteht aus einem eindeutigen Schlüssel und einem zugeordneten Wert. Man spricht in diesem Zusammenhang auch von einem Key- Value-Paar (Key-Value-Pair). Über den Schlüssel wird auf den Wert zugegriffen. In der Regel werden für den Schlüssel Strings verwendet. Es können aber auch andere Typen verwendet werden. Wichtig ist nur, dass die Werte des Schlüssels einzigartig sind und sich nicht verändern. Ein Dictionary besteht aus einer Reihe solcher Paare, welche ungeordnet abgelegt sind. Paare innerhalb eines Dictionary werden durch ein Komma getrennt, die Werte in einem Key-Value-Paar durch einen Doppelpunkt.

Grundlegende Operationen

Anlegen von einem Dictionary

In Python wird ein Dictionary erzeugt, indem man Paare zwischen zwei geschweiften Klammern {} einfügt oder die Built-in-Funktion „dict()“ verwendet. Einige Beispiele für das Anlegen eines Dictionary sind im Nachfolgenden aufgeführt.

Beispiele für das Anlegen von einem Dictionary

```
</> # leeres Dictionary
d = {}
# Dictionary mit Integer-Schlüsselwerten
d = {1: "Berlin", 2: "München", 3: "Hamburg"}
# Dictionary mit String-Schlüsselwerten
d = {"Maier": 18, "Müller": 23}
# mit dict ein Dictionary anlegen
d = dict({"Maier": 18, "Müller": 23})
```

Lesen und Schreiben von Werten

Die Werte der einzelnen Paare können nun über den Schlüsselwert angesprochen und verändert werden.

Beispiele für das Lesen und Schreiben von einzelnen Werten über den Schlüsselwert

```
</> d = {1: "Berlin", 2: "München", 3: "Hamburg"}
print(d[2])      # Ausgabe München
d[2] = "Frankfurt"
print(d[2])      # Ausgabe Frankfurt
```

Hinzufügen von neuen Werten

Neue Werte werden einem Dictionary hinzugefügt, indem man einen noch nicht verwendeten Schlüssel hinter dem Namen des Dictionary angibt und diesem einen Wert zuweist. Wird ein schon vorhandener Schlüssel verwendet, dann wird der alte Wert überschrieben.

Beispiel für das Hinzufügen von neuen Werten zu einem Dictionary

```
</> d = {1: "Berlin", 2: "München", 3: "Hamburg"}
d[4] = "Leipzig"
print(d) # {1: 'Berlin', 2: 'München', 3: 'Hamburg', 4: 'Leipzig'}
```

Löschen von Werten

Einen Wert löschen kann man mit dem Befehl „del“. Dazu muss der Schlüssel des Paares angegeben werden, welches gelöscht werden soll.

Beispiel für das Löschen von Werten

```
</> d = {1: "Berlin", 2: "München", 3: "Hamburg"}
del d[2]
print(d) # Ausgabe {1: 'Berlin', 3: 'Hamburg'}
```

Mit diesen grundlegenden Operationen kann man mit dem Dictionary schon ordentlich arbeiten. Es gibt aber noch eine Reihe von weiteren Funktionen, welche das Arbeiten mit dem Dictionary erleichtern. Einige wichtige davon sind in der nachfolgenden Tabelle aufgeführt.

Weitere Funktionen für die Arbeit mit dem Dictionary	
<code>len(d)</code>	liefert die Anzahl der Paare vom Dictionary „d“
<code>d.clear()</code>	entfernt aus dem Dictionary „d“ alle Werte
<code>d.copy()</code>	erzeugt eine Kopie vom Dictionary „d“
<code>d.pop(key)</code>	gibt den Wert des Schlüssels „key“ zurück und löscht anschließend das Paar aus dem Dictionary „d“
<code>d.setdefault(key, value)</code>	führt <code>d[key] = value</code> aus, wenn der Schlüssel „key“ noch nicht vorhanden ist
<code>d.update(d2)</code>	fügt ein Dictionary „d2“ zu „d“ hinzu und überschreibt ggf. die Werte bereits vorhandener Schlüssel

✓ Kompetenzcheck

- Geben Sie an, welche Aussagen richtig sind.
 - Eine ArrayList gehört zu den Standarddatenstrukturen in Python.
 - Mithilfe von Sets können in Python Mengenoperationen ausgeführt werden.
 - In einem Tupel kann kein Element mehrfach vorkommen.
 - Um Listen in Python benutzen zu können, muss das Modul „list“ in das Programm eingebunden werden.
 - Listen in Python können nur eine feste Anzahl von Elementen enthalten.
 - In mehrdimensionalen Listen in Python müssen alle Elemente den gleichen Datentyp besitzen.
 - Mithilfe der Funktion „del“ kann man sowohl in Listen als auch im Dictionary Elemente löschen.
 - Auf die Werte von einem Dictionary kann über den Schlüssel zugegriffen werden.
 - Mithilfe von `mitarbeiter.clear()` werden alle Elemente aus dem Dictionary „mitarbeiter“ entfernt.
- Implementieren Sie das Fallbeispiel 1 mithilfe einer mehrdimensionalen Liste in Python. Entwickeln Sie dazu eine GUI- oder Konsolenanwendung, in der die Anzahl der Reihen und die Anzahl der Sitze pro Reihe eingegeben werden können. Verwenden Sie den Zufallszahlengenerator in Python, um eine zufällige Sitzplatzbelegung zu erzeugen. Stellen Sie im Anschluss die Sitzplatzbelegung auf dem Bildschirm grafisch dar.
- Implementieren Sie das Fallbeispiel 2 mithilfe einer eindimensionalen Liste in Python. Entwickeln Sie eine GUI- oder Konsolenanwendung, in der Bestellungen hinzugefügt bzw. gelöscht werden können. Bestellungen sollen als Objekte zu speichern sein, welche die Bezeichnung des Getränks und die Anzahl enthalten. Die jeweils aktuell zu bearbeitende Bestellung soll auf dem Bildschirm angezeigt werden. Wird diese als bearbeitet markiert, dann wird sie aus der Liste gelöscht und die nächste Bestellung wird angezeigt.
- Implementieren Sie das Fallbeispiel 3 mithilfe eines Dictionary in Python. Entwickeln Sie eine GUI- oder Konsolenanwendung, mit deren Hilfe Sie die Daten eingeben und abspeichern können. Die einzelnen Daten einer Farbe werden in einem Farbbjekt hinterlegt. Durch die Eingabe der RAL-Farbnummer sollen die einzelnen Daten der entsprechenden Farbe auf dem Bildschirm angezeigt werden.
- Implementieren Sie weitere Programme mit den entsprechenden Datenstrukturen in Python, indem Sie die Aufgabe 3 der Lernsituation 2 im Arbeitsbuch bearbeiten.
- Überprüfen Sie zur Prüfungsvorbereitung Ihr Wissen im Bereich Datenstrukturen mithilfe der Aufgabe 4 der Lernsituation 2.

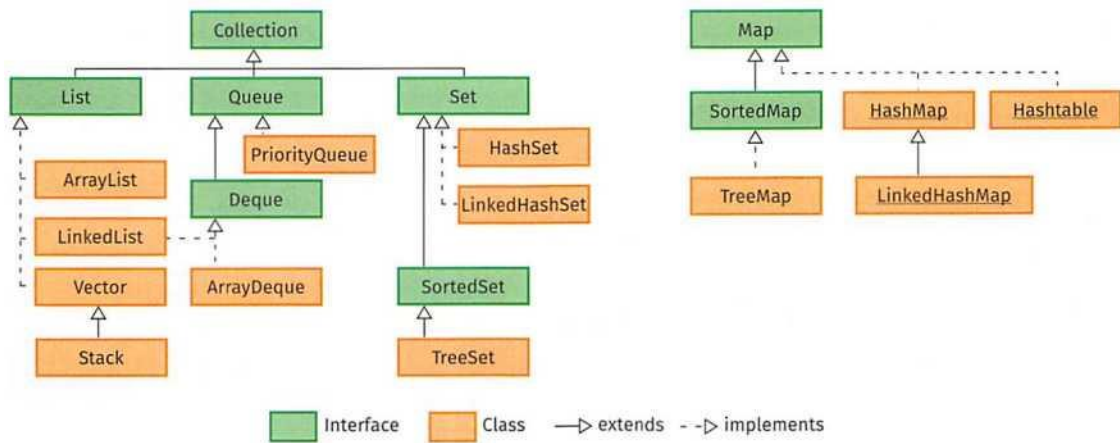
2.4. Grundlegende Datenstrukturen von Java unterscheiden und einsetzen

➤ Sie sollen grundlegende Datenstrukturen in Java kennen und anwenden können.

Java bietet eine umfangreiche Sammlung von Datenstrukturen, welche von einfachen Feldern bis hin zu Maps reichen. Die bereitgestellten Datenstrukturen sind sehr leistungsfähig und bieten eine Vielzahl von Funktionen.

Im Java-Paket „*java.util*“ sind alle Datenstrukturen von Java zu finden. Der Hauptteil der Datenstrukturen ist im Collection-Framework von Java definiert und implementiert. Eine Übersicht über das Collection-Framework mit seinen Interfaces und Klassen ist in der nachfolgenden Übersicht zu finden.





Hier werden zunächst durch Interfaces Operationen für verschiedene Datenstrukturen (Container) festgelegt. Die daraus abgeleiteten Klassen implementieren die erforderlichen Operationen und ergänzen sie durch weitere spezielle Operationen für die jeweilige Klasse.

Da es zahlreiche Datenstrukturen in Java gibt, wird hier nur näher auf Datenstrukturen eingegangen, welche sehr oft zum Einsatz kommen. Diese werden im Anschluss vertieft vorgestellt.

Grundlegende Datenstrukturen in Java	
Array	Ein Array in Java ist eine sequenzielle Sammlung von Elementen. Dabei müssen alle Elemente vom gleichen Datentyp sein und eine feste Größe aufweisen. Die Größe des Feldes kann sich während der Laufzeit nicht verändern.
ArrayList	Eine ArrayList implementiert Listen-Funktionalität durch die Abbildung auf ein Feld. Der Unterschied zu einem Array liegt darin, dass sich die Größe des Feldes während der Laufzeit ändern kann, d. h. die ArrayList kann schrumpfen und wachsen.
HashMap	Eine weitere wichtige Datenstruktur in Java ist die HashMap. In einer HashMap werden Elemente in Form von Schlüssel-Wert-Paaren abgespeichert. Auf ein einzelnes Element einer HashMap kann über einen eindeutigen Schlüssel zugegriffen werden. Wie eine ArrayList kann eine HashMap während der Laufzeit beliebig wachsen und schrumpfen.
HashSet	Ein HashSet implementiert die Schnittstelle „Set“ durch ein schnelles Hashverfahren. Die Elemente werden in einer Hashtabelle gespeichert, wodurch ein schneller Zugriff möglich ist. Im Unterschied zur HashMap und ArrayList kann in einem HashSet ein Element nicht mehrmals vorkommen.

(1) Array

Ein Array ist eine Sammlung von Elementen, welche alle den gleichen Datentyp besitzen. Auf die einzelnen Elemente eines Arrays kann über einen Index zugegriffen werden. Das Array besitzt außerdem eine festgelegte unveränderbare Größe, welche in dem Attribut „length“ gespeichert wird. Neben eindimensionalen Arrays ist es auch möglich, mehrdimensionale Arrays anzulegen. Dabei ist die Anzahl der Dimensionen nicht begrenzt.

Nachfolgend wird anhand eines zweidimensionalen Arrays exemplarisch die Erzeugung und Verwendung von mehrdimensionalen Arrays erläutert.

Zweidimensionales (mehrdimensionales) Array

Index: von 0 bis Anzahl Elemente - 1				
	0	1	2	3
	10	20	30	40

Index Zeile		0	1	2	Index Spalte
	0	13.5	-2.0	1.3	
	1	0.0	91.8	0.0	

Grundlegende Operationen

Anlegen von Arrays

Arrays in Java werden mit einem Datentyp deklariert, dem alle Elemente des Arrays entsprechen müssen. Danach folgen gewöhnlich eckige Klammern und der Bezeichner des Arrays. Da es sich bei einem Array um einen komplexen Datentyp handelt, benötigt man bei der Erzeugung des Arrays den „new“-Operator.

Beispiele für das Anlegen von Arrays

```
</> int[] a = new int[4]; // Alle Werte werden mit 0 initialisiert

int[] a1 = {10, 20, 30, 40}; // Erzeugen des Arrays a1 mit
                             // Anfangswerten

double[][] a2 = new double[2][3]; // zweidimensionales Array
double[][] a3 = {             // zweidimensionales Array mit
    {1.0, 2.0, 3.0},           // unterschiedlicher Anzahl an Elementen
    {4.0, 5.0, 6.0, 9.0},     // pro Zeile
    {7.0},
};
```

Lesen und Schreiben von Werten

Der Zugriff auf die einzelnen Elemente erfolgt wieder über die Angabe der entsprechenden Indizes. Bezogen auf ein zweidimensionales Array müssen der Index der Zeile und der Index der Spalte angegeben werden. Der Index der Zeilen bzw. der Spalten beginnt immer mit dem Wert null. Der letzte Index ist immer die Anzahl der Zeilen bzw. die Anzahl der Spalten minus eins. Wird ein Index verwendet, dessen Wert außerhalb des Arraybereichs liegt, wird ein Laufzeitfehler erzeugt.

Beispiele für den Zugriff auf Arrayelemente

```
</> System.out.println(a1[0]); // Ausgabe: 10

int wert = a1[1] + a1[2];
System.out.println(wert); // Ausgabe: 50

a2[1][2] = 70.88;
System.out.println(a2[1][2]); // Ausgabe: 70.88
```

Einfügen und Löschen von Elementen

Da die Arraygröße zur Laufzeit nicht geändert werden kann, können auch keine Elemente hinzugefügt bzw. gelöscht werden. Die Arraygröße kann durch das Attribut „length“ festgestellt werden.

Bestimmen der Größe von einem Array

```
</> System.out.println(a1.length); // Ausgabe: 4
System.out.println(a3.length); // Ausgabe: 3 (Zeilen)
System.out.println(a3[0].length); // Ausgabe: 3 (Werte)
System.out.println(a3[1].length); // Ausgabe: 4 (Werte)
System.out.println(a3[2].length); // Ausgabe: 1 (Wert)
```

(2) ArrayList

In Java muss die Größe eines Arrays am Anfang festgelegt werden. Sobald die Größe eines Arrays deklariert ist, ist es schwierig, diese zu ändern. Um dieses Problem zu beheben, kann eine ArrayList verwendet werden. Diese ermöglicht es, in der Größe veränderbare Arrays zu erstellen. Im Gegensatz zu Arrays kann eine ArrayList ihre Kapazität automatisch anpassen, wenn wir Elemente hinzufügen oder daraus entfernen.

Grundlegende Operationen

Anlegen einer ArrayList

Um eine ArrayList benutzen zu können, muss zunächst das Package „java.util.ArrayList“ importiert werden. Zunächst werden die ArrayList deklariert und der Datentyp zugewiesen, welche die Werte der ArrayList haben. Die ArrayList selbst wird danach mit dem Schlüsselwort „new“ erzeugt.

Beispiele für das Anlegen einer ArrayList

```
</> import java.util.ArrayList;
ArrayList<String> sprachen = new ArrayList<>(); // ArrayList mit
// Stringwerten
ArrayList<Double> messwerte = new ArrayList<>(); // ArrayList mit
// Doublewerten
```

Hinzufügen und Löschen von Elementen

Um der ArrayList ein Element hinzuzufügen, wird die Methode „add()“ verwendet. Zum Löschen eines Elementes wird die Methode „remove(index)“ verwendet. Dabei wird der Index des zu löschenden Elements übergeben.

Beispiel für das Hinzufügen und Löschen von Elementen

```
</> sprachen.add("Java");
sprachen.add("C");
sprachen.add("Python");
System.out.println(sprachen); // Ausgabe: [Java, C, Python]
sprachen.remove(1);
System.out.println(sprachen); // Ausgabe: [Java, Python]
```

Lesen und Schreiben von Werten

Um Werte einer ArrayList zu lesen oder zu schreiben, werden die Methoden „get(index)“ und „set(index, value)“ verwendet.

Beispiel für das Lesen und Schreiben von Werten

```
</> System.out.println(sprachen.get(0)); // Ausgabe: Java

sprachen.set(1, "Python3");
System.out.println(sprachen); // Ausgabe [Java, Python3]
```

Mit diesen grundlegenden Operationen kann man mit einer ArrayList bereits ordentlich arbeiten. Es gibt aber eine Reihe weiterer Methoden, welche das Arbeiten mit der ArrayList erleichtern. Einige wichtige davon sind in der nachfolgenden Tabelle aufgeführt.

Weitere Methoden für die Arbeit mit einer ArrayList	
a.clear()	entfernt alle Elemente aus der ArrayList „a“
a.size()	gibt die Anzahl der Elemente der ArrayList „a“ zurück
a.clone()	erstellt eine Kopie der ArrayList „a“
a.isEmpty()	ermittelt, ob die ArrayList „a“ leer ist und gibt einen Boolean-Wert zurück

(3) HashMap

Eine HashMap speichert die Elemente in Schlüssel-Wert-Paaren. Hier sind Schlüssel eindeutige Bezeichner, die verwendet werden, um auf die einzelnen Werte zuzugreifen. Die Schlüssel müssen dabei eindeutig sein. Wird versucht, einen schon verwendeten Schlüssel neu einzufügen, dann wird stattdessen der Wert des schon vorhandenen Schlüssels ersetzt. HashMaps sind im Gegensatz zu TreeMaps nicht sortiert.

Grundlegende Operationen

Anlegen von HashMaps

Um eine HashMap benutzen zu können, muss zunächst das Package „java.util.HashMap“ importiert werden. Zunächst werden die HashMap deklariert und der jeweilige Datentyp für den Schlüssel und den Wert zugewiesen. Die HashMap selbst wird danach mit dem Schlüsselwort „new“ erzeugt.

Beispiele für das Anlegen von HashMaps

```
</> import java.util.HashMap;
HashMap<Integer, String> mitarbeiter = new HashMap<>();
```

Einfügen und Löschen von Elementen

Um der HashMap ein Element hinzuzufügen, wird die Methode „put(key, value)“ verwendet. Dabei werden als Parameter der Schlüssel und der dazugehörige Wert übergeben. Zum Löschen eines Elementes kommt die Methode „remove(key)“ zum Einsatz. Dabei wird der Schlüssel des zu löschenden Elementes übergeben.

Beispiel für das Einfügen und Löschen von Elementen

```
</> mitarbeiter.put(555, "Maier"); //Hinzufügen von Elementen
mitarbeiter.put(100, "Müller");
mitarbeiter.put(788, "Schulze");
System.out.println(mitarbeiter); //Ausgabe: {100=Müller, 788=Schulze,
//555=Maier}

mitarbeiter.remove(788); //Löschen von Elementen
System.out.println(mitarbeiter); //Ausgabe: {100=Müller, 555=Maier}
```

Lesen und Schreiben von Werten

Um Werte einer HashMap zu lesen bzw. zu schreiben, werden die Methoden „get(key)“ und „replace(key, value)“ verwendet. Der Zugriff auf die entsprechenden Elemente erfolgt dabei über den Schlüssel, welcher als Parameter übergeben wird.

Beispiel für das Lesen und Schreiben von Werten

```
</> System.out.println(mitarbeiter.get(100)); // Ausgabe: Müller
mitarbeiter.replace(100, "Winkler");
System.out.println(mitarbeiter); //Ausgabe: {100=Winkler, 555=Maier}
```

Mit diesen grundlegenden Operationen kann man mit einer ArrayList bereits ordentlich arbeiten. Es gibt aber noch eine Reihe weiterer Methoden, welche einem das Arbeiten mit der ArrayList erleichtern. Einige wichtige davon sind in der nachfolgenden Tabelle aufgeführt.

Weitere Methoden für die Arbeit mit einer HashMap	
m. clear()	entfernt alle Elemente aus der HashMap „m“
m.size()	gibt die Anzahl der Elemente der HashMap „m“ zurück
m.clone()	erstellt eine Kopie der HashMap „m“
m.isEmpty()	ermittelt, ob die HashMap „m“ leer ist und gibt einen Boolean-Wert zurück.

✓ Kompetenzcheck

1. Geben Sie an, welche Aussagen richtig sind.
 - a. Java stellt nur sehr wenige Datenstrukturen zur Verfügung.
 - b. Die meisten Datenstrukturen von Java sind in dem Collection-Framework definiert.
 - c. Die Klasse „Vector“ ist vom Interface „List“ abgeleitet.
 - d. Die Klasse „TreeSet“ ist unsortiert.
 - e. Die Größe eines Arrays in Java kann während der Laufzeit schrumpfen und wachsen.
 - f. Alle Elemente eines Arrays müssen vom gleichen Datentyp sein.
 - g. Die Größe einer ArrayList in Java kann während der Laufzeit schrumpfen und wachsen.
 - h. Mit der Methode „put(value)“ werden der ArrayList neue Werte hinzugefügt.
 - i. Eine HashMap darf als Schlüssel nur Stringwerte haben.
 - j. Mit der Methode „remove(key)“ werden Elemente aus einer HashMap entfernt.

2. Implementieren Sie das Fallbeispiel 1 (S. 144) mithilfe eines zweidimensionalen Arrays in Java. Entwickeln Sie dazu eine GUI- oder Konsolenanwendung in der die Anzahl der Reihen und die Anzahl der Sitze pro Reihe eingegeben werden können. Verwenden Sie den Zufallszahlengenerator von Java, um eine zufällige Sitzplatzbelegung zu erzeugen. Stellen Sie im Anschluss die Sitzplatzbelegung auf dem Bildschirm grafisch dar.
3. Implementieren Sie das Fallbeispiel 2 (S. 145) mithilfe einer ArrayList in Java. Entwickeln Sie eine GUI- oder Konsolenanwendung, mit der Bestellungen hinzugefügt bzw. gelöscht werden können. Bestellungen sollen als Objekte zu speichern sein, welche die Bezeichnung des Getränks und die Anzahl enthalten. Die jeweils aktuell zu bearbeitende Bestellung soll auf dem Bildschirm angezeigt werden. Wird diese als bearbeitet markiert, wird sie aus der Liste gelöscht und die nächste Bestellung wird angezeigt.
4. Implementieren Sie das Fallbeispiel 3 (S. 145) mithilfe einer HashMap in Java. Entwickeln Sie eine GUI- oder Konsolenanwendung, mit deren Hilfe Sie die Daten eingeben und abspeichern können. Die einzelnen Daten einer Farbe werden in einem Farbobjekt hinterlegt. Durch die Eingabe der RAL-Farbnummer sollen die einzelnen Daten der entsprechenden Farbe auf dem Bildschirm angezeigt werden.
5. Implementieren Sie weitere Programme mit den entsprechenden Datenstrukturen in Java, indem Sie die Aufgabe 3 der Lernsituation 2 im Arbeitsbuch bearbeiten.
6. Überprüfen Sie zur Prüfungsvorbereitung Ihr Wissen im Bereich Datenstrukturen mithilfe der Aufgabe 4 der Lernsituation 2.

3. Funktionalität von Softwarekomponenten planen und implementieren

- **Sie sollen die Funktionalität von modularen Softwarekomponenten mithilfe von grafischen Beschreibungsmitteln planen und in einer Programmiersprache umsetzen.**

Die funktionalen Anforderungen für das System und dementsprechend auch für einzelne Module ergeben sich aus den funktionalen Anforderungen, welche in der Systemspezifikation festgelegt wurden. Bei dem Beispiel des „Yachthafen Resorts“ ist beispielsweise eine funktionale Anforderung an das Modul „Personalmanagement“ die Mitarbeiterverwaltung. Hieraus ergeben sich dann die funktionalen Anforderungen für die Klasse „Mitarbeiter“ usw.



Die Funktionalität, welche es umzusetzen gilt, ergibt sich aber nicht ausschließlich aus den funktionalen Anforderungen, sondern es spielen durchaus auch nicht funktionale Anforderungen (siehe auch Jahrgangsband 1, Lernfeld 5) eine Rolle. In vielen Fällen reicht es nämlich nicht aus, dass das zu entwickelnde System seine Funktion erfüllt, sondern es spielt auch eine Rolle, wie effizient es dabei ist. Es kann beispielsweise erwünscht sein, dass Datenbankabfragen nicht mehr als drei Sekunden benötigen, um Nutzern die benötigten Daten zur Verfügung zu stellen. In diesem Fall kann es nötig sein, SQL-Abfragen bezüglich der Geschwindigkeit zu optimieren, obwohl sie bereits das richtige Ergebnis liefern. Dieses Kapitel beschäftigt sich deswegen mit einer Bandbreite an Themen. Diese reichen von der Planung von Algorithmen mithilfe von Beschreibungssprachen bis hin zur Umsetzung von diesen in einer Programmiersprache. Dabei werden auch spezielle Algorithmen und deren Vor- und Nachteile näher besprochen. Des Weiteren wird die Arbeit mit Daten und Datenquellen vertieft.

3.1. Algorithmen entwerfen und implementieren

- **Sie sollen verschiedene Algorithmen mithilfe von Modellierungssprachen planen und diese dann in eine Programmiersprache implementieren.**

Dem Begriff „Algorithmus“ begegnet man heutzutage sehr häufig. In vielen Fällen wird er mit den Begriffen „Programm“ oder sogar „Software“ synonym verwendet. Dabei gibt es jedoch einige Unterschiede.

Ein Algorithmus ist erst einmal nichts weiter als eine Handlungsvorschrift, mit der man ein Problem lösen kann. Die Handlungsvorschrift selbst besteht aus einer Abfolge von klar definierten Schritten oder Regeln, die befolgt werden müssen, um ein vorher festgelegtes Ergebnis zu erhalten.

In der Regel werden Algorithmen entwickelt, um dem Menschen direkt oder indirekt die Arbeit zu erleichtern. Dazu wird der Algorithmus auf verschiedene Weise umgesetzt, z.B. in einem elektrischen



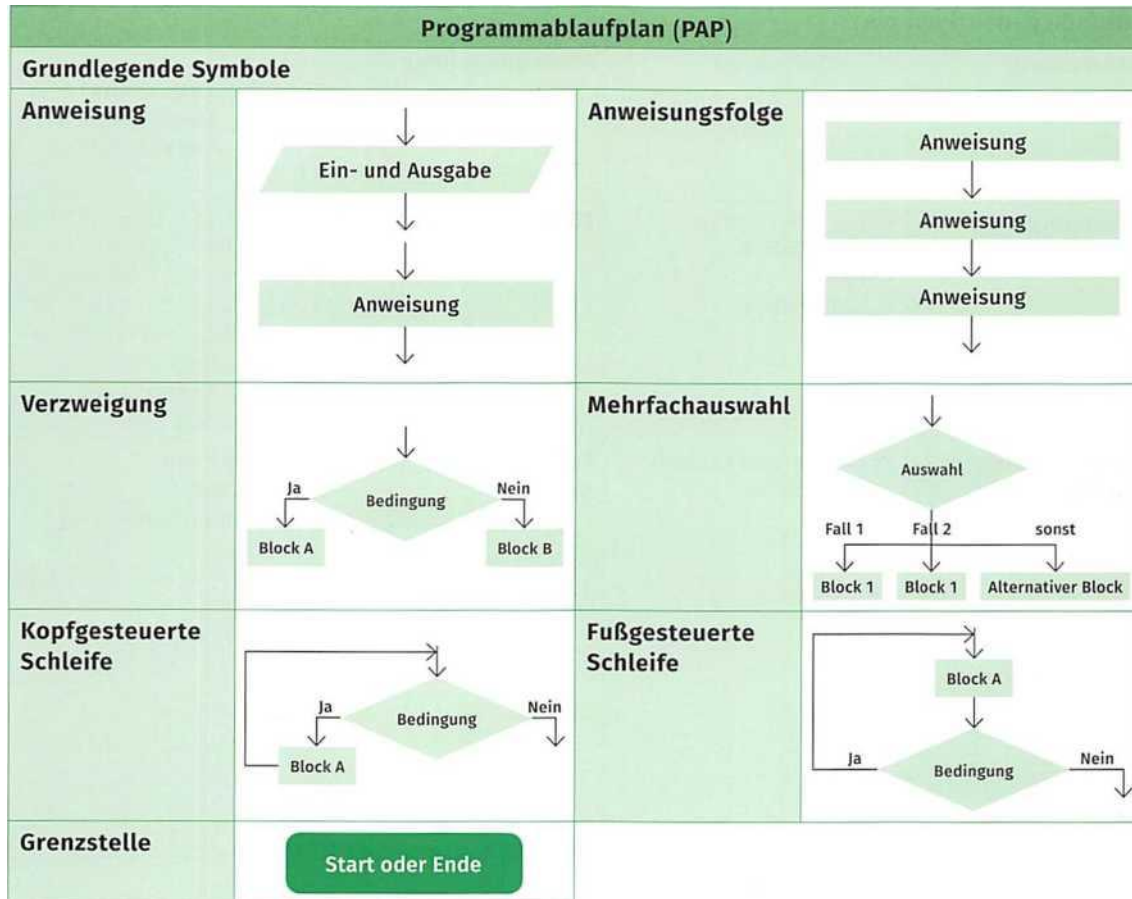
Schaltkreis oder in einem mechanischen Gerät. Die meisten Algorithmen werden heutzutage aber mithilfe von Programmiersprachen in Form von Programmen implementiert. Einfache Algorithmen, z. B. die Addition zweier Zahlen, können sofort in einer Programmiersprache umgesetzt werden und bedürfen keiner gesonderten Planung. Komplexere Algorithmen hingegen sollten vor ihrer Umsetzung modelliert werden. Eine solche Modellierung beschreibt und dokumentiert den Algorithmus programmiersprachenunabhängig.

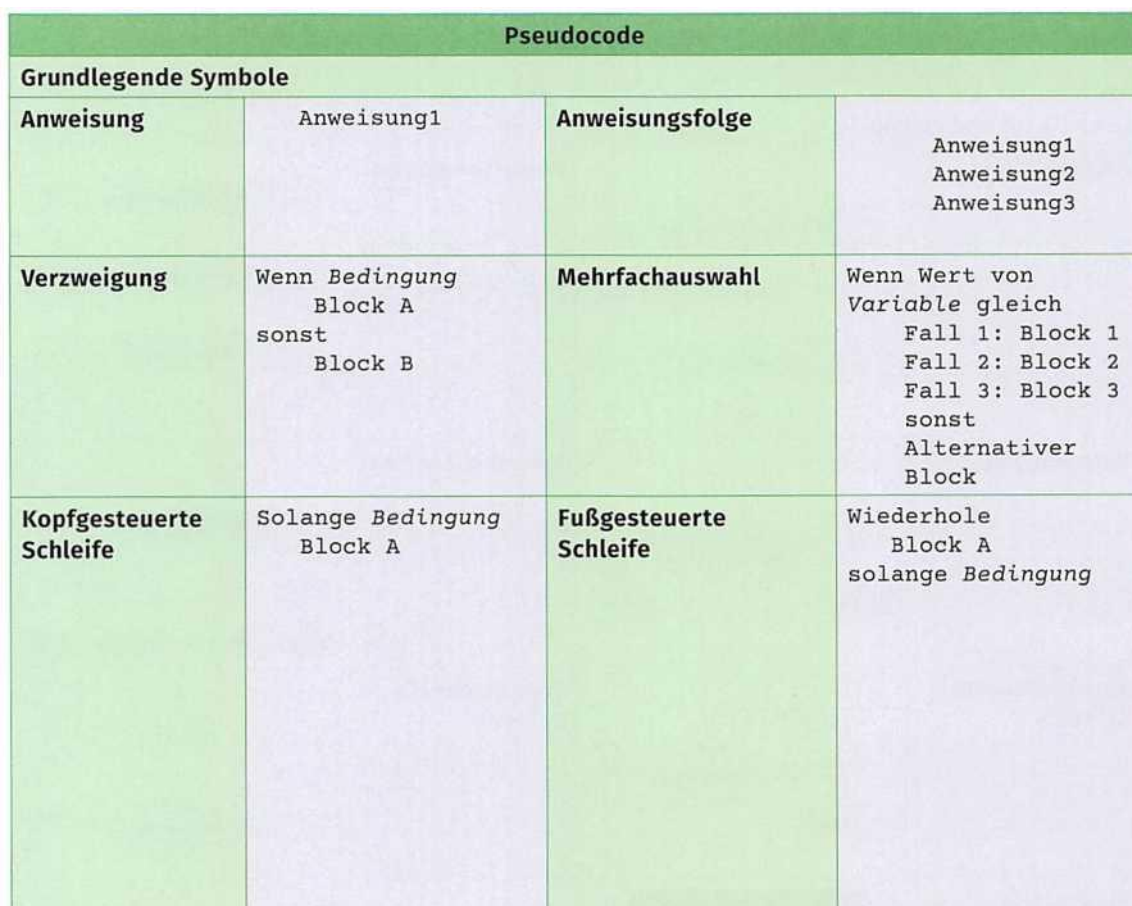
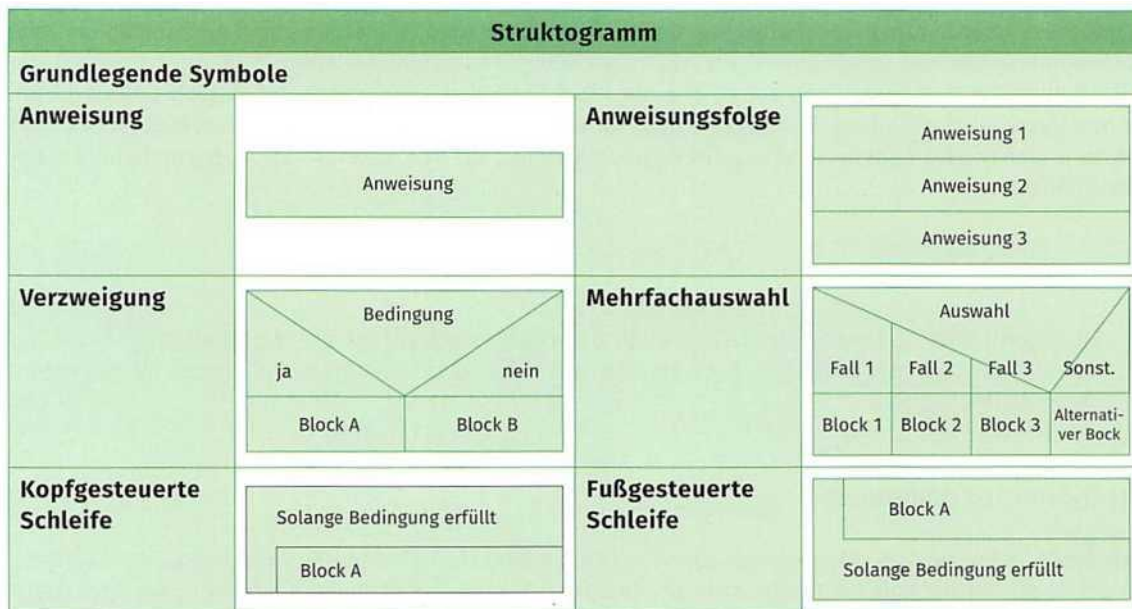
Algorithmus

Ein Algorithmus ist eine definierte und endliche Vorgehensweise, mit der ein Problem gelöst werden kann. Er besteht aus Anweisungen, welche Schritt für Schritt und in der definierten Reihenfolge abgearbeitet werden.

(1) Algorithmen modellieren

Für die Modellierung von Algorithmen können verschiedene Modellierungssprachen verwendet werden. In der Regel werden dafür entweder ein PAP, ein Struktogramm, ein Aktivitätsdiagramm oder auch Pseudocode eingesetzt. Die folgenden Tabellen geben einen Überblick über die grundlegenden Symbole dieser Modellierungssprachen.





Exemplarisch soll für folgende Ausgangslage für das Softwareprojekt „Yachthafen Resort“ ein Algorithmus modelliert und danach in Java und Python implementiert werden.

Softwareprojekt „Yachthafen Resort“ Liegeplatzverwaltung



Im Rahmen des Projektes wird eine Softwarekomponente für die Verwaltung von Liegeplätzen entwickelt. In diesem Zusammenhang soll ein Algorithmus entworfen und implementiert werden, welcher die aktuelle Auslastung der Liegeplätze des „Yachthafen Resort“ in Prozent ermittelt und anzeigt.

Die aktuellen Belegungsdaten befinden sich in einem Array „liegeplatzbelegung“, welches für die einzelnen Liegeplätze einen Boolean-Wert enthält.

Dabei entspricht:

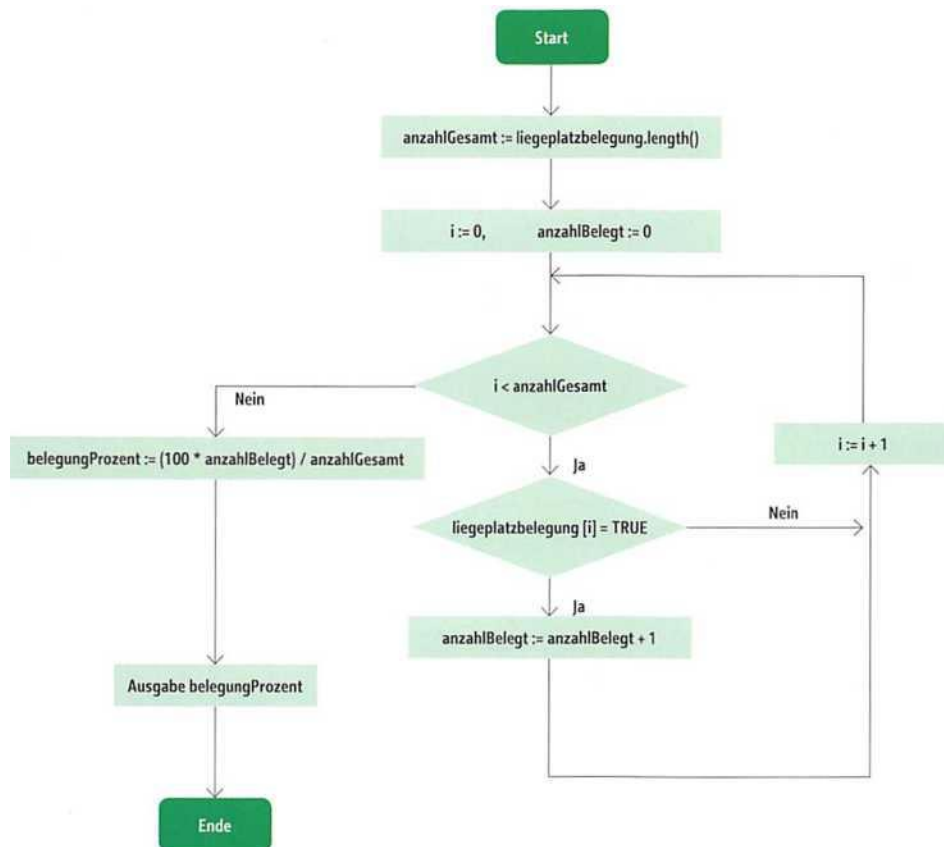
True = belegt
und
False = nicht belegt

Die Anzahl der Liegeplätze entspricht der Anzahl der Elemente des Arrays.

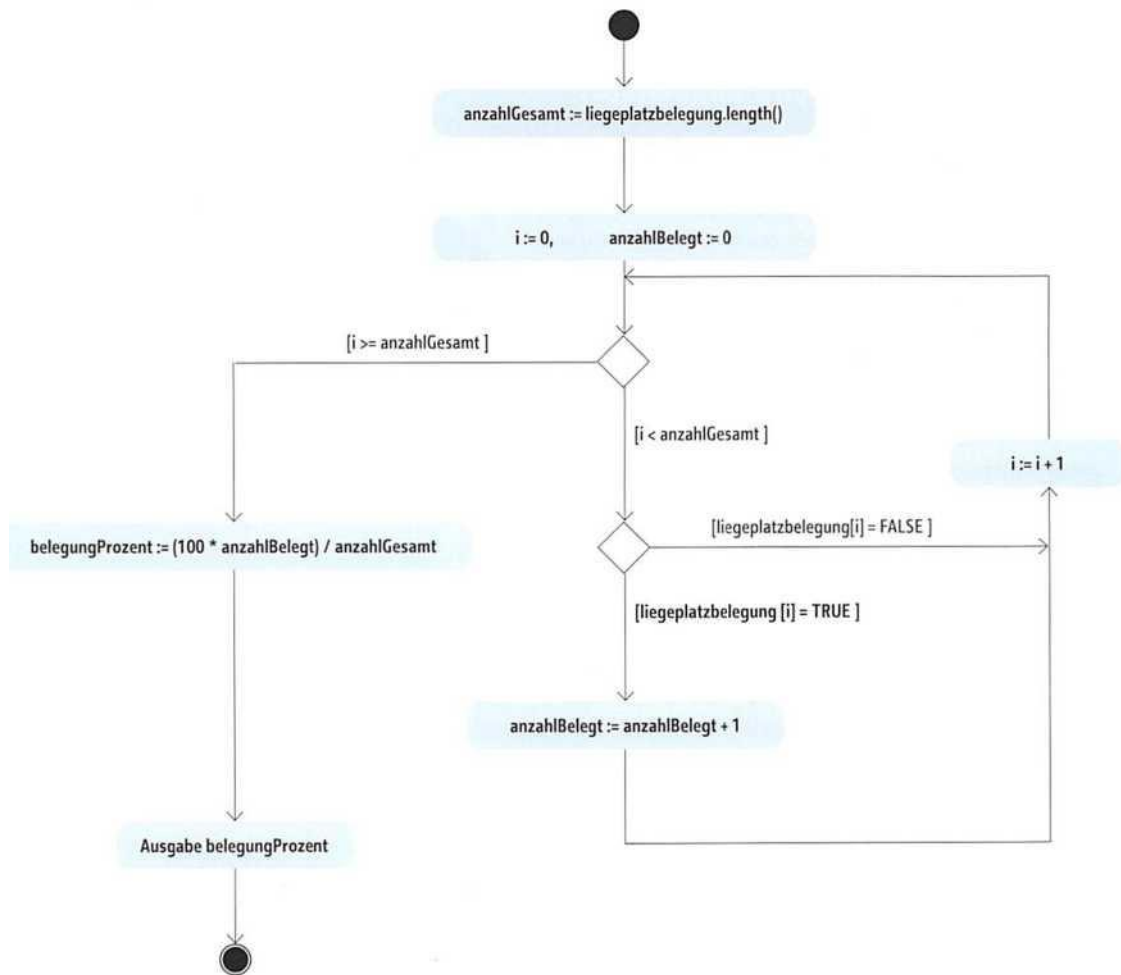


Zunächst wird der Algorithmus mithilfe einer Modellierungssprache entworfen. Vergleichend wird die Lösung mithilfe von vier verschiedenen Modellierungssprachen dargestellt.

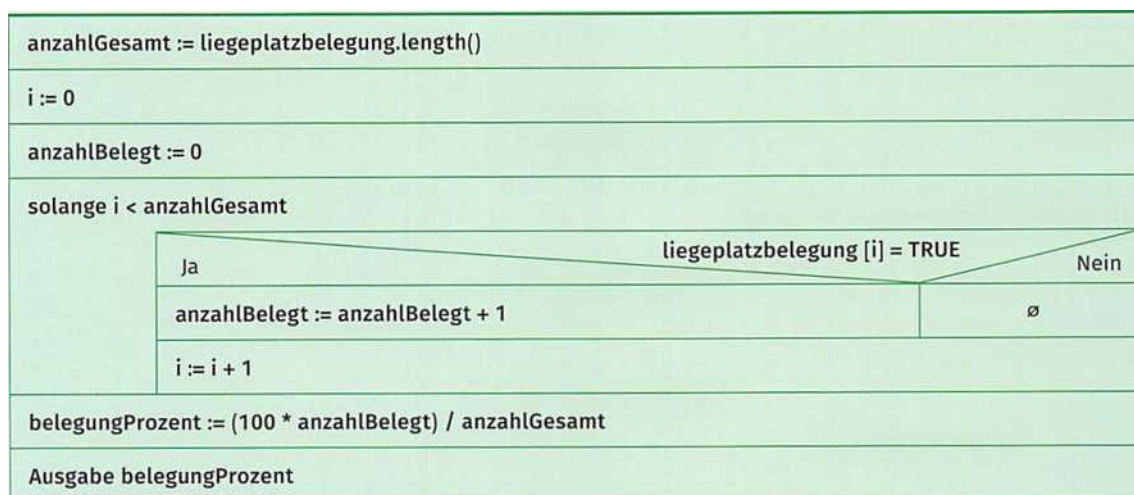
Programmablaufplan (PAP)



Aktivitätsdiagramm



Struktogramm



Pseudocode

```

</> anzahlGesamt := liegeplatzbelegung.length()
      i := 0
      anzahlBelegt := 0

      Solange i < anzahlGesamt
          Wenn liegeplatzbelegung[i] = TRUE
              anzahlBelegt := anzahlBelegt + 1
              i := i + 1

      belegungProzent := (100 * anzahlBelegt) / anzahlGesamt
      Ausgabe belegungProzent
  
```

(2) Algorithmen implementieren

Nachdem ein Algorithmus modelliert wurde, kann er nun in eine Programmiersprache implementiert werden. In den meisten Fällen kann die Modellierung eins zu eins übernommen werden. Aber es kommt auch immer wieder vor, dass die jeweilige Programmiersprache keine dem Algorithmus äquivalente Kontrollstruktur, z. B. eine Von-bis-Zähl-schleife, aufweist. In diesem Fall muss der Algorithmus entsprechend angepasst werden. Auch die Ein- und Ausgabe, die Datenstrukturen usw. müssen eventuell in der programmtechnischen Umsetzung verändert werden. Weiterhin können natürlich auch programmiersprachenbedingte Optimierungen vorgenommen werden, aber der Grundcharakter des Algorithmus sollte immer erhalten bleiben und nicht allzu stark von der Planung abweichen.

Bei den beiden nachfolgenden Lösungen wurde vor dem Algorithmus noch ein Array „liegeplatzbelegung“ mit fünf Werten initialisiert.

Umsetzung in Python

```

</> liegeplatzbelegung = [True, False, True, False, False]

anzahlGesamt = len(liegeplatzbelegung)
i = 0
anzahlBelegt = 0

while i < anzahlGesamt:
    if liegeplatzbelegung[i] == True:
        anzahlBelegt = anzahlBelegt + 1
    i = i + 1

belegungProzent = (100 * anzahlBelegt) / anzahlGesamt
print(belegungProzent)
  
```

Umsetzung in Java

```

</> public class Programm {

    public static void main(String[] args) {
        boolean liegeplatzbelegung[] = { true, false, true, false, false };
        int anzahlGesamt = liegeplatzbelegung.length;
        int i = 0;
        int anzahlBelegt = 0;

        while (i < anzahlGesamt) {
            if (liegeplatzbelegung[i] == true) {
                anzahlBelegt = anzahlBelegt + 1;
            }
            i = i + 1;
        }
        double belegungProzent = (100 * anzahlBelegt) / anzahlGesamt;
        System.out.println(belegungProzent);
    }
}
  
```

✓ Kompetenzcheck

Entwickeln Sie für die Aufgaben 1 und 2 jeweils einen Algorithmus und stellen Sie diesen zunächst mithilfe einer Modellierungssprache dar. Implementieren Sie im Anschluss den Algorithmus in einer Programmiersprache.

- Der Algorithmus der Beispielaufgabe auf S. 163 soll um eine Monatsauswertung der Liegeplatzbelegung erweitert werden. Dazu ist anstatt eines eindimensionalen Arrays mit der aktuellen Tagesbelegung ein zweidimensionales Array mit den Tagesbelegungen eines Monats gegeben.

Beispiel:

	0	1	2	3	4	5	...	n
0	true	false	true	false	true	false	...	true
1	false	true	true	true	true	true	...	false
...	...	...	...	...	...	...	...	...
n	false	false	true	false	false	false	...	false

Index Liegeplatz

Index Tag:
Der erste Tag
des Monats hat
den Wert 0.

Es sollen folgende Daten ermittelt und ausgegeben werden:

- Prozentuale Liegeplatzbelegung pro Tag
 - Durchschnittliche Liegeplatzbelegung in dem Monat
 - Der oder die Tage mit der höchsten bzw. der niedrigsten Belegung
 - Anzahl der Tage, welche eine Belegung von 80% und höher ausweisen.
- Für mehrere Module der Software des „Yachthafen Resorts“ ist ein Algorithmus für das Erstellen und das Testen einer Prüfnummer erforderlich. In diesem Zusammenhang soll u.a. ein Algorithmus entwickelt werden, welcher aus einer positiven ganzen Zahl die Quersumme der Ziffern dieser Zahl berechnet und ausgibt.

Beispiel:

Zahl: 2311
Berechnung Quersumme: $2 + 3 + 1 + 1 = 7$

- Planen und implementieren Sie verschiedene Algorithmen für das „Yachthafen Resort“, indem Sie die Aufgabe 1 der Lernsituation 3 im Arbeitsbuch bearbeiten.
- Bearbeiten Sie zur Prüfungsvorbereitung die Aufgabe 2 der Lernsituation 3 im Arbeitsbuch.

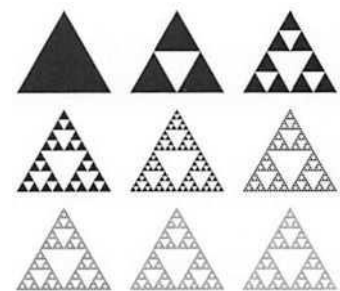
3.2. Rekursive Algorithmen umsetzen

➤ Sie sollen einfache rekursive Algorithmen planen und in einer Programmiersprache umsetzen.

Rekursion ist ein spezielles Verfahren, mit dem ein komplexes Problem auf ein einfacheres zurückgeführt und auf diese Weise gelöst wird. Einfache Algorithmen, z.B. die Berechnung einer quadratischen Gleichung, das Lesen einer Datei oder das Senden einer Nachricht, erfordern keine Rekursion. Algorithmen, welche rekursiv ausgeführt werden können, sollten in der Regel folgende Merkmale aufweisen.

Merkmale von Algorithmen, welche rekursiv ausgeführt werden können

- Der Algorithmus muss mehrmals ausgeführt werden.
- Der Algorithmus benötigt Daten, die sich jedes Mal ändern.
- Der Algorithmus bleibt gleich und ändert sich nicht.
- Es gibt eine Bedingung, bei deren Erfüllung der Algorithmus endet.



Allerdings müssen diese Merkmale nicht immer vollständig zutreffen. Es gibt z. B. rekursive Algorithmen, welche keine Abbruchbedingung haben. Diese unendlichen Rekursionen haben auch ihren eigenen Anwendungsbereich, sind aber nicht Gegenstand der nachfolgenden Betrachtungen.

Rekursiver Algorithmus

Generell ruft ein rekursiver Algorithmus sich selbst auf und übergibt einen Rückgabewert als Parameter an den neuen Aufruf. Oder anderes ausgedrückt: Das Ergebnis einer Rekursion ist die Eingabe für die nächste Rekursion. Um nicht in einer Endlosschleife zu landen, muss der Algorithmus eine Bedingung enthalten, unter der kein weiterer rekursiver Aufruf stattfindet.

Steht der Ablauf vom Algorithmus fest, dann muss dieser programmtechnisch umgesetzt werden. Dazu wird eine

Funktion geschrieben, welche einen oder mehrere Übergabeparameter und einen Rückgabewert besitzt. Diese Funktion ruft sich dann selbst auf.

Beispielaufgabe

Als Beispiel soll hier ein Algorithmus für die Umrechnung einer positiven ganzen Dezimalzahl in eine Dualzahl nach der Divisionsmethode (siehe auch Jahrgangsband 1, Lernfeld 5) vorgestellt werden. Dabei wird eine ganzzahlige Division durchgeführt. Bei jedem Divisionsergebnis ohne Rest wird eine 0 und bei einem Ergebnis mit Rest eine 1 notiert. Das ganzzahlige Divisionsergebnis ist dann die nächste Zahl, mit der der Algorithmus wieder durchlaufen wird. Dieses wird solange wiederholt, bis nur noch eine dezimale 0 übrig bleibt. Der Ablauf wird in einem Beispiel mit der Zahl 10 dargestellt.

Beispiel:	10_{10}	→	1010_2	
	Ganz		Rest	
10 : 2 =	5		0	niedrigstes Bit
5 : 2 =	2		1	
2 : 2 =	1		0	
1 : 2 =	0		1	höchstes Bit
1				Abbruchbedingung erreicht

Als nächstes wird untersucht, ob der Algorithmus sich zur Rekursion eignet.

Muss der Algorithmus mehrmals ausgeführt werden?	in der Regel ja
Benötigt der Algorithmus Daten, die sich jedes Mal ändern?	ja, eine positive ganze Zahl
Bleibt der Algorithmus immer derselbe?	ja
Gibt es eine Bedingung, bei deren Erfüllung der Algorithmus endet?	ja, wenn die positive ganze Zahl den Wert 0 hat

Aufgrund der Antworten kann der Algorithmus rekursiv gestaltet werden. Eine mögliche Umsetzung in Python und Java ist nachfolgend aufgeführt.

Umsetzung in Python

```
</> def decimal_to_binary(number):
    if number < 0:
        return 'Es muss eine positive ganze Zahl sein!'
    n = number // 2 # Ganzzahldivision
    r = number % 2 # ganzzahliger Rest 0 oder 1
    if n == 0: # Abbruchbedingung für die Rekursion
        return str(r)
    else:
        return decimal_to_binary(n) + str(r) # rekursiver
                                                # Funktionsaufruf

number = int(input('Geben Sie eine positive ganze Zahl:'))
print(decimal_to_binary(number))
```

Umsetzung in Java

```

import java.util.Scanner;

public class Programm {
    public static void main(String[] args) {
        System.out.print("Geben Sie eine positive ganze Zahl: ");
        Scanner sc = new Scanner(System.in);
        int number = sc.nextInt();
        System.out.println(decimalToBinary(number));
    }

    static String decimalToBinary(int number) {
        if (number < 0) {
            return "Es muss eine positive ganze Zahl sein!";
        }
        int n = number / 2; // Ganzzahldivision
        int r = number % 2; // ganzzahliger Rest 0 oder 1
        if (n == 0) { // Abbruchbedingung für die Rekursion
            return Integer.toString(r);
        } else {
            // rekursiver Funktionsaufruf
            return decimalToBinary(n) + Integer.toString(r);
        }
    }
}

```

Alle rekursiven Algorithmen lassen sich auch durch iterative Lösungen darstellen. Eine iterative Lösung ist meist effizienter und benötigt weniger Speicherplatz. Eine rekursive Lösung hingegen ist in der Regel eleganter, übersichtlicher und kommt mit weniger Quellcode aus, weil z. B. auf Hilfsvariablen und Schleifenzähler verzichtet werden kann.

Trotz aller Eleganz sollte man sich jederzeit der Anfälligkeit für Stack Overflows (Überläufe) von rekursiven Lösungen bewusst sein. Es ist wichtig, sich das Ende der Rekursion (Abbruchbedingung) genau zu überlegen. Der Grund dafür ist, dass die wiederholten Funktionsaufrufe mit allen zwischengespeicherten Werten auf dem Stapelspeicher (Stack) abgelegt werden. Hier kann die Rekursion zu einem Stack Overflow führen, wenn sie nicht abgebrochen wird, was in der Regel das Programm abstürzen lässt.

✓ Kompetenzcheck

- Gegeben ist nebenstehender Algorithmus, welcher die Summe der ganzen Zahlen von 0 bis n bildet. Entwickeln Sie für die iterative eine rekursive Lösung. Stellen Sie diese zunächst als Struktogramm dar. Machen Sie einen Schreibtischtest, in dem Sie alle Programmschritte mit den entsprechenden Werten für $n = 3$ aufschreiben. Implementieren Sie Ihre Lösung im Anschluss in einer Programmiersprache.
- Bearbeiten Sie im Arbeitsbuch die Lernsituation 3 mit der Aufgabe 3.

3.3. Sortiervverfahren beschreiben, implementieren und anwenden

- **Sie sollen einfache Sortieralgorithmen beschreiben und implementieren können. Des Weiteren sollen Sie von den Programmiersprachen bereitgestellte Sortierfunktionen in Ihren Programmen einsetzen.**

Eingabe: n
summe := 0
i := 1
solange i <= n
summe := summe + i
i := i + 1
Ausgabe: summe

Die Sortierung von Datenmengen ist ein grundlegendes Thema in der Informatik. Vor allem bei großen Datenmengen ist ein effizienter Algorithmus notwendig. Diese wurden mit der Zeit entwickelt und stehen in Form von Bibliotheken zur Verfügung, sodass diese nicht neu implementiert werden müssen. Um ein grundlegendes Verständnis für die Belange der Sortierung von Datenmengen zu bekommen, bietet es sich jedoch an, einfache Sortieralgorithmen zu durchdenken und selbst zu implementieren.



Sortieralgorithmus

Ein Sortieralgorithmus ist ein Verfahren, durch das die Daten einer linearen Datenstruktur, z.B. einem Array, nach einem bestimmten Kriterium geordnet werden. Die Eingabe ist z.B. ein Array mit einer bestimmten Anzahl an Werten (z. B. Zahlen, Buchstaben usw.) und eventuell ein Sortierkriterium (z.B. aufsteigend, absteigend). Die Ausgabe ist dann das sortierte Array. Das Ziel ist es dabei nicht bloß, die Daten zu sortieren, sondern das Sortieren auch so effizient wie möglich zu gestalten. Die Effizienz der Sortieralgorithmen ist in den meisten Fällen vom Ausgangszustand der Daten abhängig. Sind z.B. die Daten schon sortiert, brauchen die meisten Sortieralgorithmen weniger Zeit. Deswegen wird bei der Betrachtung der Laufzeit immer zwischen einem Best Case, einem Average Case und einem Worst Case unterschieden.

Beispiele für Sortieralgorithmen:

Bubblesort, Heapsort, Insertionsort, Mergesort, Quicksort, Selectionsort, Shellsort

Als Beispiel wird nachfolgend der Bubblesort näher betrachtet.

(1) Bubblesort

Der Bubblesort ist ein einfacher Sortieralgorithmus, der benachbarte Elemente eines Arrays wiederholt tauscht, wenn sie sich in falscher Reihenfolge befinden, und so das Array sortiert. Je nachdem, ob aufsteigend oder absteigend sortiert werden soll, wandert das größte bzw. kleinste Element wie eine „Bubble“ durch das Feld, bis es am Ende angekommen ist. Dort angekommen wird es als sortiert betrachtet und braucht nicht mehr im nächsten Durchlauf berücksichtigt zu werden. Dann wiederholt sich der Vorgang. Der Ablauf ist beendet, wenn kein Tausch erfolgt ist oder (AnzahlArrayElemente -1)-Durchläufe durchgeführt wurden.

Beispielaufgabe

Der Ablauf wird in der folgenden Darstellung nochmal verdeutlicht. Dabei wird ein Array mit fünf Werten vorgegeben und es soll aufsteigend sortiert werden, d.h., es werden immer dann zwei benachbarte Elemente getauscht, wenn das erste Element größer als das zweite Element ist. Nach dem zweiten Durchlauf ist in dem Beispiel das Array eigentlich schon sortiert. Allerdings braucht der Algorithmus noch einen Durchlauf ohne Tausch, damit er informiert ist, dass das Array sortiert ist. Dieses ist nach dem dritten Durchlauf der Fall. Da dort kein Tausch stattgefunden hat, ist das Array sortiert und der Algorithmus kann abgebrochen werden.

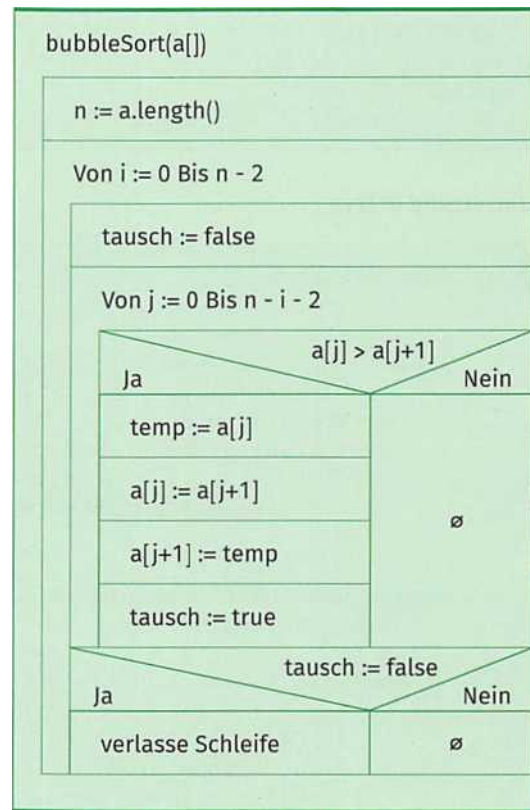
Eingangsarray:	65	21	42	38	55
Erster Durchlauf:					
65 ↔ 21	42	38	55	Tausch: 65 und 21	
21	65 ↔ 42	38	55	Tausch: 65 und 42	
21	42	65 ↔ 38	55	Tausch: 65 und 38	
21	42	38	65 ↔ 55	Tausch: 65 und 55	
21	42	38	55	65	Ende erster Durchlauf
Zweiter Durchlauf:					
21	42	38	55	65	Tausch: nein
21	42 ↔ 38	55	65	Tausch: 42 und 38	
21	38	42	55	65	Tausch: nein
21	38	42	55	65	Ende zweiter Durchlauf
Dritter Durchlauf:					
21	38	42	55	65	Tausch: nein
21	38	42	55	65	Tausch: nein
21	38	42	55	65	Ende dritter Durchlauf

Der Algorithmus für den Bubblesort ist nachfolgend mithilfe eines Struktogramms abgebildet. Er stellt eine mögliche Lösung dar. Hier wurden als Schleifen Zählschleifen verwendet. Es ist aber durchaus möglich, andere Schleifen zu wählen.

Bei den Zählschleifen in einem Struktogramm ist zu beachten, dass diese eine „Bis“-Bedingung zum Abbruch enthalten, während vielen Programmiersprachen eine „Solange“-Abbruchbedingung eigen ist. Dieses wirkt sich auf die Formulierung der Abbruchbedingung aus. Im Struktogramm muss deswegen von der Anzahl der Feldelemente 2 abgezogen werden, während es in der Implementierung in Python oder Java reicht, 1 abzuziehen. Bei der inneren Schleife wird zusätzlich noch der aktuelle Wert von i abgezogen. Dadurch wird erreicht, dass die schon sortierten Elemente am Ende nicht mehr verglichen werden.

Falls nicht getauscht wurde, wird im Struktogramm nur notiert, dass die Schleife verlassen wird. Der Befehl und die konkrete Umsetzung können von Programmiersprache zu Programmiersprache verschieden ausfallen.

Der Algorithmus kann nun in einer Programmiersprache implementiert werden.



Umsetzung in Python

```

</> # Bubble Sort in Python

def bubbleSort(a):
    n = len(a)
    for i in range(0, n-1):
        tausch = False
        for j in range(0, n-i-1):
            if a[j] > a[j+1]:
                temp = a[j]
                a[j] = a[j+1]
                a[j+1] = temp
                tausch = True
        if tausch == False:
            break

array = [65, 21, 42, 38, 55]
bubbleSort(array)
print ("Sortiertes Feld: ")
print (array)
  
```

Umsetzung in Java

```
// Bubble Sort in Java

public class Programm {
    public static void main(String[] args) {
        int array[] = {65, 21, 42, 38, 55};
        bubbleSort(array);
        System.out.println("Sortiertes Array: ");
        for(int i = 0; i < array.length; i++)
            System.out.print(array[i] + " ");
    }

    static void bubbleSort(int a[]) {
        int n = a.length;
        for (int i = 0; i < n-1; i++) {
            boolean tausch = false;
            for (int j = 0; j < n-i-1; j++) {
                if (a[j] > a[j+1]) {
                    int temp = a[j];
                    a[j] = a[j+1];
                    a[j+1] = temp;
                    tausch = true;
                }
            }
            if (tausch == false) {
                break;
            }
        }
    }
}
```

(2) Sortierfunktionen verwenden

In der Regel liefern moderne Programmiersprachen schon vorimplementierte Sortierfunktionen mit, welche man benutzen kann. Nachfolgend ein Beispiel in Python bzw. Java.

Sortieren von Arrays mithilfe von vorgegebenen Sortierfunktionen

Hier wird jeweils ein Array mit Zahlen und eins mit Wörtern initialisiert und danach sortiert. Das Array mit den Zahlen wird standardmäßig aufsteigend und das mit den Wörtern absteigend sortiert. Der Quellcode ist selbsterklärend. Die Ausgabe ist in beiden Fällen die gleiche und nebenstehend abgebildet.

```
[12, 4, -7, 71, 8, -4]
['wald', 'baum', 'wasser', 'meer']
[-7, -4, 4, 8, 12, 71]
['wasser', 'wald', 'meer', 'baum']
```

Umsetzung in Python

```
zahlen = [12, 4, -7, 71, 8, -4]
woerter = ["wald", "baum", "wasser", "meer"]
print(zahlen)
print(woerter)
zahlen.sort()
woerter.sort(reverse=True)
print(zahlen)
print(woerter)
```

Umsetzung in Java

```
</> import java.util.Arrays;
import java.util.Collections;
public class Program {
    public static void main(String[] args) {
        int[] zahlen = { 12, 4, -7, 71, 8, -4};
        String[] woerter = {"wald", "baum", "wasser", "meer"};
        System.out.println(Arrays.toString(zahlen));
        System.out.println(Arrays.toString(woerter));
        Arrays.sort(zahlen);
        Arrays.sort(woerter, Collections.reverseOrder());
        System.out.println(Arrays.toString(zahlen));
        System.out.println(Arrays.toString(woerter));
    }
}
```

✓ Kompetenzcheck

1. Der Bubblesort-Algorithmus von S. 169 soll so erweitert werden, dass zusätzlich ermittelt wird, wie oft ein Tausch stattgefunden hat. Dieser Wert soll ausgegeben werden.
2. Entwerfen und implementieren Sie, auch in Vorbereitung auf die Prüfung, weitere einfache Sortieralgorithmen, indem Sie die Aufgabe 4 der Lernsituation 3 im Arbeitsbuch bearbeiten.

3.4. Komprimierungsverfahren beschreiben, implementieren und anwenden

- **Sie sollen einfache Komprimierungsalgorithmen beschreiben und implementieren können. Des Weiteren sollen Sie bereits implementierte Komprimierungsverfahren in Ihren Programmen einsetzen.**

Speicherplatz war schon immer ein wertvolles Gut in der Informatik. Besonders bei großen Datenmengen besteht ein erhöhtes Interesse daran, die Größe soweit wie möglich zu reduzieren, ohne dass ein Verlust an den benötigten Informationen auftritt. Hierzu wurden über die Jahre Algorithmen entwickelt, welche diese Aufgabe übernehmen. Auch hier gilt, dass diese in Form von Bibliotheken zur Verfügung stehen.



Die nachfolgend vorgestellte Lauflängenkodierung dient als Einstieg in diesen Bereich.

Komprimierungsalgorithmus

Ein Komprimierungsalgorithmus ist ein Verfahren, welches die Menge von digitalen Daten reduziert. Dabei wird u.a. eine Reduzierung des benötigten Speicherplatzes erreicht oder die benötigte Zeit für eine Datenübertragung wird verkürzt. Es gibt verschiedene Komprimierungsalgorithmen für die unterschiedlichsten Arten von Datenmengen, z.B. die Komprimierung von Grafik-, Audio oder Videodaten.

Jeder Komprimierungsalgorithmus versucht zunächst, redundante Informationen aus der Datenmenge zu beseitigen und die Informationen effizienter zu speichern. In diesen Fällen spricht man auch von verlustfreier Komprimierung, weil alle Originalinformationen wiederhergestellt werden können. Verlustbehaftete Komprimierung reduziert die Datenmenge noch weiter, indem sie alle nicht benötigten Informationen zusätzlich aus der Datenmenge entfernt, z.B. alle Frequenzinformationen, welche nicht zum Hörbereich des Menschen gehören. Allerdings lassen sich dann aus der komprimierten Datenmenge nicht mehr alle Originalinformationen herstellen.

Beispiele für Komprimierungsalgorithmen:

JPEG, GIF, TIFF, WMA, MP3, WMV, MPEG

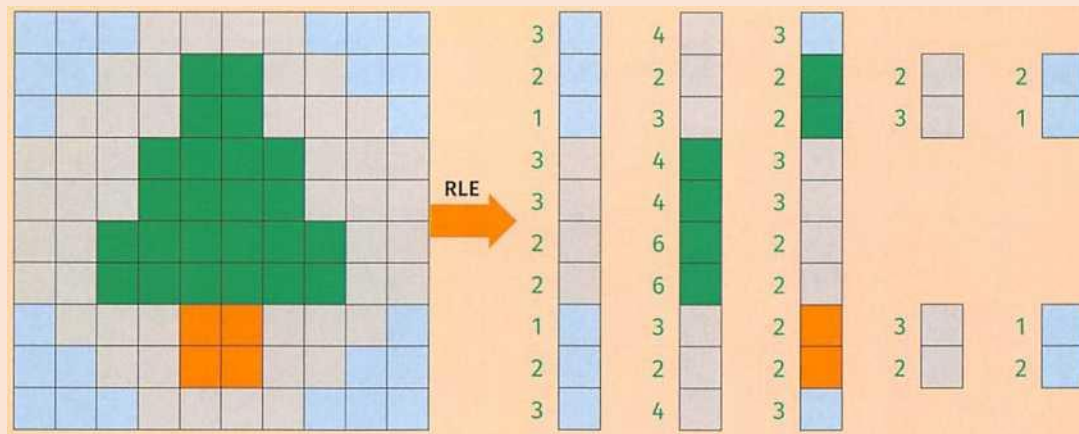
(1) Lauflängenkodierung

Der Algorithmus der Lauflängenkodierung (run-length encoding - RLE) dient der verlustfreien Datenkomprimierung. Es ist ein sehr einfacher Algorithmus und wird vor allem dazu verwendet, Daten zu komprimieren, bei denen identische Datenwerte sehr oft hintereinander austreten. Dazu wird jede Sequenz von identischen Werten durch deren Anzahl und den eigentlichen Wert ersetzt. Diese Vorgehensweise ist am effizientesten bei Daten, welche viele solcher Sequenzen enthalten, z. B. einfache grafische Bilder, welche wenige Farben und großen Farbflächen aufweisen. In diesen Fällen wird eine hohe Kompression erreicht. Wie alle Komprimierungsalgorithmen gibt es aber auch beim RLE Fälle, bei denen keine Verringerung der Datenmenge erreicht wird oder sich die Datenmenge sogar vergrößert. Dieses muss auch der Fall sein, da man sonst eine Datenmenge beliebig oft komprimieren könnte, bis sie nur noch 1 Bit groß ist. Das ist natürlich nicht möglich. Beim RLE vergrößert sich die Datenmenge beispielsweise, wenn sich viele unterschiedliche Datenwerte ständig abwechseln, wie es beispielsweise bei Fotos der Fall ist. Die Lauflängenkodierung wird heutzutage als Vorkodierungsschritt verwendet, z.B. bei der Bildkompression von JPEG oder TIFF, um sich

im folgenden Kodierungsschritt Aufwand zu sparen.

Beispielaufgabe

Es liegt eine Rastergrafik (siehe auch Jahrgangsband 1, Lernfeld 5) vor, welche unkomprimiert pro Pixel 3 Byte benötigt (RGB). Das würde idealisiert eine Datenmenge von $10 \cdot 10 \cdot 3 \text{ Byte} = 300 \text{ Byte}$ ergeben. Zudem ist dargestellt, wie die Daten nach der Komprimierung mit der Längenlaufkodierung gespeichert würden.



Ausgehend davon, dass eine ganze Zahl (im Bereich bis 0 bis 255) 1 Byte und Farbe (RGB) 3 Byte benötigt, ergibt sich für die komprimierte Datenmenge folgender idealisierter Wert:

$$(38 \cdot 1 \text{ Byte}) + (38 \cdot 3 \text{ Byte}) = 38 \text{ Byte} + 114 \text{ Byte} = 152 \text{ Byte}$$

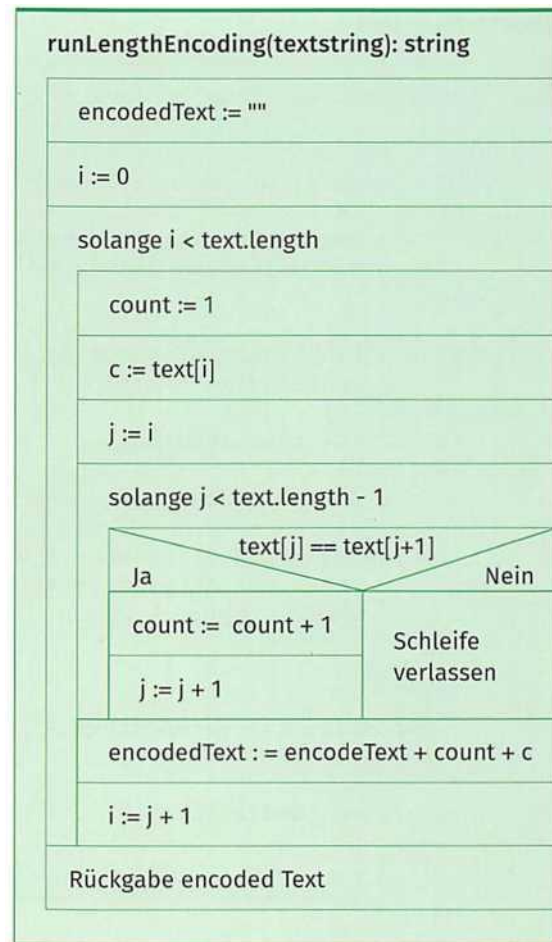
Hiermit lässt sich nun die prozentuale Größe der komprimierten Datenmenge im Vergleich zur Originaldatenmenge berechnen:

$$(152 \text{ Byte} \cdot 100 \text{ Prozent}) : 300 \text{ Byte} = 50,66 \%$$

In diesem Fall ist die zu speichernde Datenmenge im Vergleich zum Original ca. halb so groß.

Der Algorithmus für die Lauflängenkodierung ist nebenstehend als Struktogramm abgebildet. Dabei wird aber nicht wie in der Beispielaufgabe ein Rasterbild komprimiert, sondern exemplarisch nur ein Text, welcher aus verschiedenen Buchstaben besteht. In diesem Beispiel kommen wieder zwei Schleifen zum Einsatz. Diesmal wurden hier zwei kopfgesteuerte Schleifen verwendet. Es ist aber natürlich möglich, jede andere Schleifenart hier einzusetzen. Mithilfe der äußeren Schleife wird der gesamte Text einmal durchlaufen. Dabei wird sich der aktuelle Buchstabe gemerkt und mithilfe der inneren Schleife gezählt, wie oft er hintereinander vorkommt. Wird ein neues Zeichen gefunden, wird die innere Schleife sofort verlassen. Danach werden die Anzahl und der Buchstabe dem komprimierten Text hinzugefügt. Für folgenden Text „aaaaabbbccccccdefff“ würde als komprimierter Text „5a3b7c-1d1e3f“ herauskommen. Dabei ist zu beachten, dass in dem zu komprimierenden Text alle Zeichen bis auf Ziffern vorkommen dürfen. Bei Ziffern wäre das Ergebnis auch richtig, könnte unter Umständen aber nicht mehr eindeutig interpretiert werden. Der Text „aaaaa555501000zz“ würde „5a451011302z“ ergeben und es kann u.a. nicht mehr eindeutig festgestellt werden, ob 45 mal 1, 451 mal 0 usw. oder doch 4 mal 5 gemeint ist.

Der Algorithmus kann nun in eine Programmiersprache implementiert werden.



Umsetzung in Python

```

</> # Lauflängenkodierung in Python

def run_length_encoding(text):
    encoded_text = ""
    i = 0
    while i < len(text):
        count = 1
        c = text[i]
        j = i
        while j < len(text) - 1:
            if text[j] == text[j + 1]:
                count = count + 1
                j = j + 1
            else:
                break
        encoded_text = encoded_text + str(count) + c
        i = j + 1
    return encoded_text

text = "aaaaabbbccccccdefff"
encoded_text = run_length_encoding(text)
print(encoded_text)
  
```

Umsetzung in Java

```
// Lauflängenkodierung in Java

public class Program {
    public static void main(String[] args) {
        String text = "aaaaabbbbccccccdefff";
        String encodedText = runLengthEncoding(text);
        System.out.print(encodedText);
    }

    static String runLengthEncoding(String text) {
        String encodedText = "";
        int i = 0;
        while (i < text.length()) {
            int count = 1;
            char c = text.charAt(i);
            int j = i;
            while (j < text.length()-1) {
                if (text.charAt(j) == text.charAt(j+1)) {
                    count = count + 1;
                    j = j + 1;
                } else { break; }
            }
            encodedText = encodedText + Integer.toString(count) + c;
            i = j + 1;
        }
        return encodedText;
    }
}
```

(2) Dateien zu ZIP-Archiven komprimieren

Auch im Falle der Komprimierung stehen viele wichtige Algorithmen in Form von Bibliotheken zur Verfügung, welche nur ins Programm eingebunden und deren Funktionen angewandt werden müssen. Nachfolgend ein Beispiel für eine Komprimierung mithilfe von ZIP. Dabei handelt es sich zunächst um ein Dateiformat, das verlustfreie Datenkomprimierung unterstützt. Eine ZIP-Datei kann eine oder mehrere Dateien oder Verzeichnisse enthalten, welche komprimiert wurden. Das ZIP-Dateiformat erlaubt eine Reihe von Komprimierungsalgorithmen, z.B. das häufig eingesetzte „Deflate“.

Deflate-Algorithmus

Der Deflate-Algorithmus ist ein Algorithmus zur verlustlosen Datenkompression, welcher ursprünglich für das ZIP-Dateiformat entwickelt wurde.

Eine Anwendung im Rahmen des Projektes „Yachthafen Resort“ wäre z. B. die Komprimierung von Bilddateien der Liegeplätze. Eine Beispielumsetzung in Python und Java ist in der nachstehenden Tabelle aufgeführt.

Komprimierung von Bilddateien mithilfe von ZIP

Python

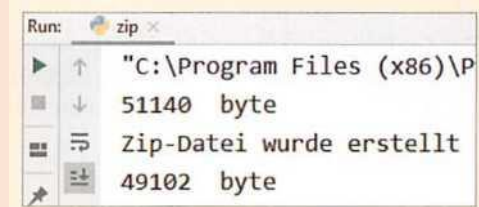
Um Dateien in Python zu einer ZIP-Datei zu packen, muss das Paket „zipfile“ in das Programm eingebunden werden. Danach wird eine ZIP-Datei angelegt und zum Schreiben geöffnet. Als Parameter wird neben dem Namen auch die Art der Komprimierung angegeben. Mit der Methode „write“ wird eine entsprechende Datei gepackt und dem ZIP-Archiv hinzugefügt.

Umsetzung in Python

```
import os, zipfile

print(os.path.getsize("Liegeplatz.jpg"), " byte")
zipfile.ZipFile('liegeplatz.zip', 'w', zipfile.ZIP_DEFLATED).
write("Liegeplatz.jpg")
print("Zip-Datei wurde erstellt")
print(os.path.getsize("Liegeplatz.zip"), " byte")
```

Zur Kontrolle wird die Größe der Datei einmal vor und einmal nach der Komprimierung ausgegeben. Dabei ist zu beachten, dass eine jpg-Datei schon eine komprimierte Datei ist.



Java

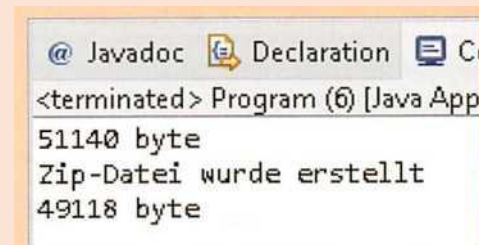
Um Dateien in Java zu einer ZIP-Datei zu packen, muss das Paket „java.util.zip.*“ in das Programm eingebunden werden. Danach wird eine ZIP-Datei angelegt und zum Schreiben geöffnet. Schließlich werden verschiedene Output- und Input-Streams angelegt und geöffnet und mit „write“ der Inhalt gepackt und in die ZIP-Datei geschrieben.

Umsetzung in Java

```
</> import java.io.*;
import java.util.zip.*;

public class Program {
    public static void main(String[] args) {
        try {
            File file = new File("Liegeplatz.jpg");
            System.out.println(file.length() + " byte");
            FileOutputStream fos = new FileOutputStream("Liegeplatz.zip");
            ZipOutputStream zos = new ZipOutputStream(fos);
            FileInputStream fis = new FileInputStream(file);
            ZipEntry zipEntry = new ZipEntry(file.getName());
            zos.putNextEntry(zipEntry);
            byte[] bytes = new byte[1024];
            int length = fis.read(bytes);
            while(length >= 0) {
                zos.write(bytes, 0, length);
                length = fis.read(bytes);
            }
            zos.close();
            fis.close();
            fos.close();
            System.out.println("Zip-Datei wurde erstellt");
            File zipfile = new File("Liegeplatz.zip");
            System.out.println(zipfile.length() + " byte");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Zur Kontrolle wird die Größe der Datei einmal vor und einmal nach der Komprimierung ausgegeben. Dabei ist zu beachten, dass eine jpg-Datei schon eine komprimierte Datei ist.



Die Dokumentation zum Paket „zipfile“ ist unter <https://docs.python.Org/3/library/zipfile.html> zu finden.

Die Dokumentation zum Paket „java.util.zip“ ist unter <https://docs.oracle.com/javase/7/docs/api/java/util/zip/package-summary.html> zu finden.

✓ Kompetenzcheck

1. Entwickeln Sie für die Lauflängenkodierung einen Algorithmus, welcher einen komprimierten Text wieder in einen Originaltext überführt. Erstellen Sie dazu zunächst ein Struktogramm und implementieren danach Ihre

Lösung.

2. Bearbeiten Sie im Arbeitsbuch die Lernsituation 3 mit der Aufgabe 5.

3.5. Verschlüsselungsverfahren beschreiben, implementieren und anwenden

- **Sie sollen einfache Verschlüsselungsalgorithmen beschreiben und implementieren können. Des Weiteren sollen Sie bereits implementierte Verschlüsselungsverfahren in Ihren Programmen einsetzen.**

Die Vertraulichkeit ist ein wichtiger Punkt im Rahmen der Informationssicherheit. Hier soll u.a. gewährleistet werden, dass Informationsinhalte nur Befugten zugänglich sind. In diesem Zusammenhang werden Verschlüsselungsalgorithmen zur Verschlüsselung von Informationsinhalten eingesetzt. Der Bedarf der Menschen an Möglichkeiten zur Sicherung von vertraulichen Informationen war schon immer sehr groß, sodass schon sehr früh Verschlüsselungsverfahren entwickelt wurden. Gleichzeitig wurde auch immer versucht, an die verschlüsselten Informationen zu gelangen. So wurden mit der Zeit immer bessere Verschlüsselungsalgorithmen notwendig, besonders weil es mit dem Beginn des Computerzeitalters immer einfacher wurde, die vorhandenen Verschlüsselungsalgorithmen zu knacken.



Verschlüsselungsalgorithmus

Ein Verschlüsselungsalgorithmus ist ein Verfahren, mit dessen Hilfe Informationen ver- und entschlüsselt werden. Dabei wird zwischen den beiden Hauptgruppen der symmetrischen und der asymmetrischen Verschlüsselung unterschieden. Beide Verfahren haben ihre Vor- und Nachteile. Heutzutage wird meist eine Kombination der beiden Verfahren angewendet. Symmetrische Verschlüsselungsverfahren verwenden zur Ver- und Entschlüsselung den gleichen Schlüssel. Das Kennzeichen von asymmetrischen Verschlüsselungsverfahren ist, dass zur Verschlüsselung ein anderer Schlüssel verwendet wird als zur Entschlüsselung. Zum Verschlüsseln wird ein „öffentlicher Schlüssel“ benutzt, zu dem jeder Zugang hat, und zum Entschlüsseln ein „privater Schlüssel“, den nur der Empfänger der Nachricht kennt.

Beispiele für Verschlüsselungsalgorithmen:

Triple DES (Data Encryption Standard), RSA, Blowfish, Twofish, AES (Advanced Encryption Standard)

Das nachfolgend vorgestellte Cäsar-Chiffre stammt aus römischer Zeit und dient nur als Einstieg in diesen Bereich.

(1) Cäsar-Chiffre

Das Cäsar-Chiffre ist ein sehr einfaches Verschlüsselungsverfahren. Dieses Verfahren ist sehr unsicher und dient hier vor allem als Einstieg in den Themenbereich der Verschlüsselung. Die Methode ist nach Julius Cäsar benannt, welcher sie zur geheimen Kommunikation verwendete. Bei dieser Technik wird jedes Zeichen von einem Klartext durch ein anderes Zeichen aus einem Zeichensatz ersetzt. Die neuen Zeichen werden ermittelt, indem man alle Zeichen des Klartextes um einen bestimmten Wert nach rechts oder links im Zeichensatz verschiebt. Dabei können verschiedene Zeichensätze zugrunde gelegt werden. In vielen Fällen wird einfach nur das Alphabet der jeweiligen Sprache genommen.

Beispielaufgabe

In diesem Beispiel wird zur Verschlüsselung der ASCII-Zeichensatz (siehe auch Jahrgangsband 1, Lernfeld 5) mit 128 Zeichen zugrunde gelegt. Die ersten 32 Zeichen und das Zeichen an der Stelle 127 eignen sich nicht zur Verschlüsselung. Das muss später bei der Entwicklung vom Algorithmus berücksichtigt werden. Hier wird nun ein Klartext jeweils um drei Zeichen nach rechts verschoben, um das neue Zeichen zu bekommen. Wenn man nun beispielsweise das große „D“ um drei Zeichen nach rechts in der ASCII-Tabelle verschiebt, dann erhält man das große „G“ (E « F « G). Das Leerzeichen ergibt z.B. die # und der Punkt die 1.

Klartext:

D	a	s		i	s	t		e	i	n		T	e	x	t	.
---	---	---	--	---	---	---	--	---	---	---	--	---	---	---	---	---

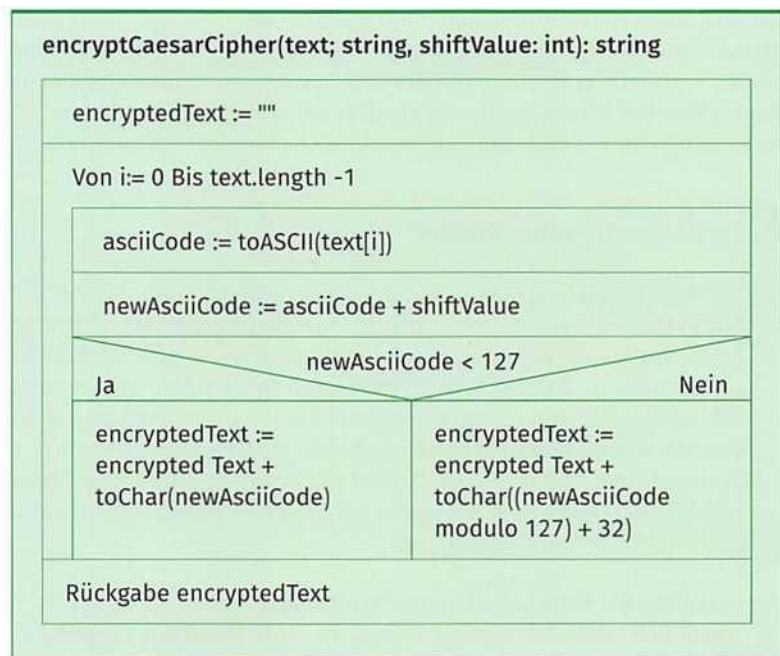
Verschiebung: 3

G	d	v	#	l	v	w	#	h	l	q	#	W	h	{	w	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Der Algorithmus für eine Umsetzung des Cäsar-Chiffre ist im nebenstehenden Struktogramm abgebildet. Hier werden zwei Funktionen verwendet:

1. toASCII(Zeichen) - gibt den ASCII-Code eines Zeichens zurück.
2. toChar(ASCII-Code) - gibt das Zeichen, welches zu einem ASCII-Code gehört, zurück.

Beim Algorithmus wird von jedem Zeichen des Klartextes der ASCII-Code ermittelt und zu diesem ein fester Wert addiert, welcher nun den ASCII-code eines neuen Zeichens darstellt. Zu beachten ist dabei, dass der Zeichensatz auf 128 Zeichen begrenzt ist. Wird diese Grenze überschritten, dann wird der Zeichensatz von vorne durchlaufen. Dieses kann durch die Verwendung von Modulo erreicht werden. Allerdings muss dann noch zu dem ermittelten Wert eine 32 dazu addiert werden, da die Zeichen 0 bis 31 Steuerzeichen im ASCII-Zeichensatz sind und nicht zur Zeichendarstellung verwendet werden. Der Algorithmus kann nun in eine Programmiersprache implementiert werden.



Umsetzung in Python

```

</> # Cäsar Chiffre in Python

def encrypt_caesar_cipher(text, shift_value):
    encrypted_text = ""
    for i in range(len(text)):
        ascii_code = ord(text[i])
        new_ascii_code = ascii_code + shift_value
        if new_ascii_code < 127:
            encrypted_text += chr(new_ascii_code)
        else:
            encrypted_text += chr((new_ascii_code % 127) + 32)
    return encrypted_text

text = "Das ist ein Text."
shift_value = 3
encrypted_text = encrypt_caesar_cipher(text, shift_value)

print("Klartext: " + text)
print("Verschiebung: " + str(shift_value))
print("Verschlüsselter Text: " + encrypted_text)
  
```

Umsetzung in Java

```
// Cäsar Chiffre in Java

public class Program {
    public static void main(String[] args) {
        String text = "Das ist ein Text.";
        int shiftValue = 3;
        String encryptedText = encryptCaesarCipher(text, shiftValue);
        System.out.println("Klartext: " + text);
        System.out.println("Verschiebung: " + shiftValue);
        System.out.println("Verschlüsselter Text: " + encryptedText);
    }

    static String encryptCaesarCipher(String text, int shiftValue) {
        String encryptedText = "";
        for(int i = 0; i < text.length(); i++) {
            int asciiCode = text.charAt(i);
            int newAsciiCode = asciiCode + shiftValue;
            if (newAsciiCode < 127) {
                encryptedText += (char)newAsciiCode;
            } else {
                encryptedText += (char)((newAsciiCode % 127) + 32);
            }
        }
        return encryptedText;
    }
}
```

(2) Vorimplementierte Verschlüsselungsalgorithmen nutzen

Auch wenn es eine gute Übung ist, Verschlüsselungsalgorithmen zu erfinden und zu implementieren, sollten alle, die nicht im Bereich der Kryptographie tätig sind und Verschlüsselung bei Kundenlösungen einsetzen wollen, folgende Regeln beachten:

1. Regel: Erfinden Sie niemals einen eigenen Verschlüsselungsalgorithmus!
2. Regel: Implementieren Sie nie einen eigenen Verschlüsselungsalgorithmus!

Bei der Entwicklung und Implementierung von eigenen Verschlüsselungsalgorithmen entstehen zu viele „Löcher“. Stattdessen sind bereits implementierte und bewährte Verschlüsselungsverfahren zu benutzen. Selbst bei den bewährten Verfahren werden noch immer Schwachstellen gefunden. Trotzdem hilft die Implementierung von Verschlüsselungsalgorithmen zu Übungszwecken dabei, die Kryptographie und die dort auftretenden Problemstellungen zu verstehen.

Für die meisten Programmiersprachen stehen verschiedene Verschlüsselungsalgorithmen in Form von Bibliotheken zur Verfügung. Nachfolgend wird die Ver- und Entschlüsselung von einem Text in Python und Java vorgestellt.

Vorimplementierte Verschlüsselungsalgorithmen

Python

Für Python stehen verschiedene kryptographische externe Bibliotheken zur Verfügung. Für das Beispiel wird „*cryptography*“ verwendet. Um diese Bibliothek nutzen zu können, muss sie zuvor mit „*pip install cryptography*“ installiert werden. Mit diesem Paket kann die Fernet-Verschlüsselung genutzt werden. Fernet selbst ist eine symmetrische Verschlüsselung, welche das 128-Bit-AES Verfahren im CBC-Modus verwendet. Zunächst muss ein Schlüssel generiert werden, welcher sowohl zum Verschlüsseln als auch zum Entschlüsseln genutzt wird.

Im nachfolgenden Beispiel wird ein Text verschlüsselt und wieder entschlüsselt und das jeweilige Ergebnis ausgegeben.

Umsetzung in Python

```
</> from cryptography.fernet import Fernet
#-----
# Schlüssel erzeugen
#-----
key = Fernet.generate_key()
cipher = Fernet(key)
plain_text = 'Eine wirklich sehr geheime Nachricht!'
print("Originaltext: " + plain_text)
#-----
# Verschlüsseln
#-----
plain_data = plain_text.encode()
encrypted_data = cipher.encrypt(plain_data)
encrypted_text = encrypted_data.decode('UTF-8')
print("Verschlüsselter Text: " + encrypted_text)
#-----
# Entschlüsseln
#-----
encrypted_data = encrypted_text.encode()
decrypted_data = cipher.decrypt(encrypted_data)
decrypted_text = decrypted_data.decode('UTF-8')
print("Entschlüsselter Text: " + decrypted_text)
```

Ausgabe

```
Originaltext: Eine wirklich sehr geheime Nachricht!
Verschlüsselter Text: gAAAAABh_h8a1B7qzzIX4CH9Gr8PH061UASTj_ct3JkbZu7mIX3Se2Hjhjnfjv6gdy_K7IPWvGwi-0i3yUAB0jiJ8KHB_TeiQa15pHRELa3gHj0t0vB7IH0R7d169pVjv--doqchngt
Entschlüsselter Text: Eine wirklich sehr geheime Nachricht!
```

Java

In Java steht mit dem Paket „*javax.crypto*“ ein Paket für die Verschlüsselung zur Verfügung. Zunächst muss wieder ein Schlüssel generiert werden, mit dem der Text ver- und entschlüsselt wird. Dazu wird ein Verschlüsselungsverfahren angegeben. In dem Beispiel wird AES verwendet. Der AES-Algorithmus ist ein symmetrisches Verschlüsselungsverfahren, welches kryptographische Schlüssel von 128, 192 und 256 Bit unterstützt. Der ver- bzw. entschlüsselte Text wird jeweils ausgegeben.

Umsetzung in Java

```

import java.util.Base64;
import javax.crypto.*;

public class Program {
    public static void main(String[] args) {
        try {
            //-----
            // Schlüssel erzeugen
            //-----
            KeyGenerator keygen = KeyGenerator.getInstance("AES");
            keygen.init(128);
            SecretKey key = keygen.generateKey();
            Cipher cipher = Cipher.getInstance("AES");
            String plainText = "Eine wirklich sehr geheime Nachricht!";
            System.out.println("Originaltext: " + plainText);
            //-----
            // Verschlüsseln
            //-----
            cipher.init(Cipher.ENCRYPT_MODE, key);
            byte[] encryptedData = cipher.doFinal(plainText.getBytes());
            String encryptedText = Base64.getEncoder().encodeToString(
                                                                    encryptedData);
            System.out.println("Verschlüsselter Text: " + encryptedText);
            //-----
            // Entschlüsseln
            //-----
            cipher.init(Cipher.DECRYPT_MODE, key);
            encryptedData = (Base64.getDecoder().decode(encryptedText));
            byte[] decryptedData = cipher.doFinal(encryptedData);
            String decryptedText = new String(decryptedData);
            System.out.println("Entschlüsselter Text: " + decryptedText);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Ausgabe

```

Originaltext: Eine wirklich sehr geheime Nachricht!
Verschlüsselter Text: cJfcNv/9T7YKKdfVJ3ieKIyktcFat3Ni8MuAl1Lf0oWbQFaVNX7vBGsqfT9jN/ag
Entschlüsselter Text: Eine wirklich sehr geheime Nachricht!

```

✓ Kompetenzcheck

1. Entwickeln Sie einen Algorithmus, welcher nach Eingabe der Verschiebung einen mit dem Cäsar- Chiffre verschlüsselten Text wieder entschlüsselt und ausgibt. Modellieren Sie zunächst den Algorithmus mithilfe eines Struktogramms und implementieren Sie diesen anschließend in einer Programmiersprache.
2. Entwerfen und implementieren Sie zur Übung weitere einfache symmetrische Verschlüsselungsalgorithmen. Bearbeiten Sie dazu die Aufgaben 6 und 7 der Lernsituation 3 im Arbeitsbuch.

3.6. Hashfunktionen beschreiben, implementieren und anwenden

- **Sie sollen einfache Hashalgorithmen beschreiben und implementieren können. Des Weiteren sollen Sie bereits implementierte Hashverfahren in Ihren Programmen einsetzen.**

Hashfunktionen werden in vielen Bereichen der Informatik eingesetzt. Ein großer Bereich ist auch in diesem Fall die Informationssicherheit. Neben der Vertraulichkeit kommt hier noch der Bereich der Integrität hinzu.

Im Gegensatz zur Verschlüsselung kann ein Hashwert nicht wieder „entschlüsselt“ oder rückgängig gemacht werden. Deswegen sind Hashverfahren auch keine Verschlüsselungsverfahren, auch wenn sie u.a. im Bereich der Kryptographie, z.B. zum Speichern von Passwörtern, eingesetzt werden.



» Hashalgorithmus

Ein Hashalgorithmus ist ein Verfahren, womit eine Zeichenfolge beliebiger Länge auf eine Zeichenfolge fester Länge reduziert wird. Das Ergebnis vom Algorithmus ist der sogenannte Hashwert, welcher aus Zahlen besteht. Oftmals wird der Hashwert auch in hexadezimaler Form angegeben, sodass die Zeichenkette des Hashwertes auch die Buchstaben a bis f enthält, z.B. „67ac4ff011efa52“. Dabei kann es vorkommen, dass für unterschiedliche Eingabewerte derselbe Hashwert erzeugt wird. In diesem Fall spricht man von einer Kollision. Durch die Beschränkung der Länge des Hashwertes ist die Menge der möglichen Hashwerte meist kleiner als die Menge der möglichen Eingaben. Daher sind solche Kollisionen meistens unvermeidlich, wenn auch selten, und es muss Verfahren zur Kollisionserkennung geben. Ein guter Hashalgorithmus zeichnet sich dadurch aus, dass er möglichst wenige Kollisionen erzeugt. Deswegen werden immer neue Hashverfahren entwickelt.

Beispiele für Hashalgorithmen:

SHA-256/224, SHA-512/384, MD5 (unsicher)

Weitere Einsatzgebiete sind: Prüfsummen, digitale Signaturen, Hashtable oder Transaktionen einer Blockchain, wie sie beispielsweise bei „Bitcoin“ zum Einsatz kommt. Es gibt etliche Hashalgorithmen. Die meisten von ihnen aber sind unsicher oder erzeugen zu viele Kollisionen.

Ein guter Hashalgorithmus sollte folgende Eigenschaften haben:

Eigenschaften von guten Hashalgorithmen	
Unidirektional	Hashalgorithmen sollten nur in einer Richtung funktionieren, d. h., es sollte aus dem Ausgangswert der Hashwert sehr leicht zu berechnen, aber aus dem Hashwert sollte nicht der Ausgangswert berechnet werden können.
Kollisionssicher	Der Hashalgorithmus sollte bei unterschiedlichen Eingaben auch unterschiedliche Hashwerte liefern. Ist diese Voraussetzung erfüllt, so spricht man auch von kryptographischen Hashfunktionen. Ansonsten sollte die Wahrscheinlichkeit von Kollisionen sehr gering sein.
Schnell	Der Hashalgorithmus muss den Hashwert sehr schnell berechnen können.

Natürlich wird auch hier immer wieder versucht, aus dem Hashwert die Originaldaten zu generieren. Bei einigen lange eingesetzten Hashfunktionen, z. B. MD5, ist dieses gelungen. Verfahren, bei denen die Herstellung der Originaldaten wieder möglich ist, gelten als unsicher und sollten nicht eingesetzt werden. Der im Anschluss vorgestellte Hashalgorithmus erfüllt nicht alle diese Eigenschaften. Aber er ist ein guter Einstieg in den Bereich der Hashalgorithmen.

(1) Beispiel für einen einfachen Hashalgorithmus

Als Eingabe sollen dem Algorithmus eine beliebige Zeichenkette und die Länge des Hashwertes zur Verfügung gestellt werden. Dieser Algorithmus verarbeitet dann jeweils Blöcke von vier Zeichen der Zeichenkette, bildet daraus ganzzahlige Werte und addiert diese dann zu einem Hashwert zusammen. Zum Schluss wird der berechnete gesamte Hashwert noch auf die übergebene Länge begrenzt. Grundsätzlich werden von den vier Zeichen eines Blocks die ASCII-Werte der Zeichen addiert. Da aber beispielsweise die Zeichenfolge „abcd“ den gleichen Wert ergibt wie „acdb“ oder „dcba“, wird jeder ASCII-Wert noch mit einer Zahl multipliziert, welche sich in jedem Durchlauf verändert. Am Ende eines Blockes erhält diese Zahl wieder ihren Ausgangswert.

Beispielaufgabe

Eingabe: Text: „passwort“, Länge: 6

Der Multiplikator wird in dem vorgestellten Algorithmus in jedem Durchlaufeines Blockes mit 64 multipliziert.

Block	Buchstabe	ASCII-Wert	Multiplikator (mul)	ASCII-Wert * mul	Gesamtsumme
1	P	112	1	112	112
1	a	97	64	6208	6320
1	s	115	4096	471 040	477360
1	s	115	262144	30146560	30623 920
2	w	119	1	119	30 624 039
2	o	111	64	7104	30 631 143
2	r	114	4096	466944	31 098087
2	t	116	262144	30408704	61 506791

Der Hashwert ist der letzte Wert der Gesamtsumme: 61 506791

Dieser muss nun noch auf eine Länge von 6 begrenzt werden. Dazu werden, z. B. mithilfe von Modulo, die letzten 6 Ziffern ermittelt.

Hashwert der Länge 6: 506 791

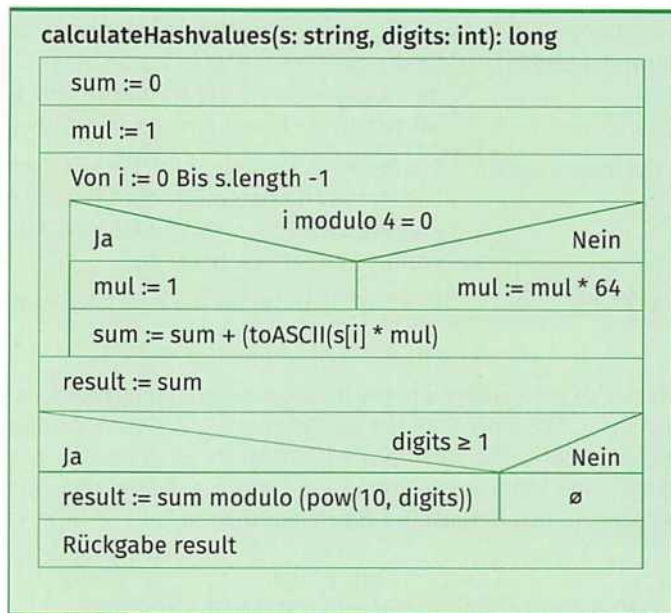
Dieser Wert stellt nun den Hashwert mit einer definierten Länge dar.

Der Algorithmus für eine Umsetzung vom Hashalgorithmus ist im nebenstehenden Struktogramm abgebildet. Hier werden zwei Funktionen verwendet

1. toASCII(Zeichen) - gibt den ASCII-Code eines Zeichens zurück.
2. pow(x, y) - berechnet x hoch y und gibt den Wert zurück.

Der Algorithmus durchläuft den gesamten übergebenen String. Es wird der ASCII-Code von jedem Zeichen ermittelt und mit einem Wert „(mul)“ multipliziert. Dieser wird bei jedem Durchlauf vergrößert. Die Blockeinteilung wird durch den Modulo-Operator erreicht. Jeden vierten Durchlauf wird der Multiplikator wieder auf 1 zurückgesetzt. Bei einer sehr langen Zeichenfolge führt die Berechnung der Summe in der Regel dazu, dass eine 32-Bit-Ganzzahl überläuft, wodurch einige der Bits höherer Ordnung verloren gehen, da die resultierenden Werte so groß sind. Dieses führt jedoch zu keinem

Problem, da das Ziel darin besteht, nur einen Hashwert zu berechnen. Zum Schluss wird die errechnete Summe noch auf die geforderte Anzahl an Ziffern reduziert. Dieses erfolgt wieder durch Modulo mit einem Vielfachen von 10. Der Wert 10 liefert beispielsweise einen einstelligen Hashwert und der Wert 10000 einen vierstelligen Hashwert. Der Algorithmus kann nun in einer Programmiersprache implementiert werden.



Umsetzung in Python

```
</>
import math
def calculate_hashvalue(s, digits):
    sum = 0
    mul = 1
    for i in range(0, len(s)):
        if i % 4 == 0:
            mul = 1
        else:
            mul = mul * 64
        sum = sum + (ord(s[i]) * mul)
    result = sum
    if digits >= 1:
        result = sum % math.pow(10, int(digits))
    return int(result)

password = "password"
length_hashvalue = 6
print(calculate_hashvalue(password, length_hashvalue))
```

Umsetzung in Java

```
</>
public class Program {
    public static void main(String[] args) {
        String password = "password";
        int length_hashvalue = 6;
        long hv = calculateHashvalue(password, length_hashvalue);
        System.out.println(hv);
    }

    static long calculateHashvalue(String s, int digits) {
        long sum = 0;
        long mul = 1;
        for (int i = 0; i < s.length(); i++) {
            if (i % 4 == 0) {
                mul = 1;
            } else {
                mul = mul * 64;
            }
            sum = sum + (s.charAt(i) * mul);
        }
        long result = sum;
        if (digits >= 1) {
            result = sum % (long)(Math.pow(10, digits));
        }
        return result;
    }
}
```

(2) SHA-256-Hashwert mit vorimplementierten Funktionen erzeugen

Der SHA (Secure Hash Algorithm) ist eine der beliebtesten kryptographischen Hashfunktionen. Ein kryptographischer Hash kann verwendet werden, um z. B. eine Signatur für einen Text oder eine Datei zu erstellen. Der SHA-256 generiert einen fast eindeutigen 256-Bit-Hash mit fester Größe, welcher wie schon erwähnt nicht auf den ursprünglichen Wert zurückgeführt werden kann. Derzeit ist SHA-256-Hashing weit verbreitet, da es als der sicherste Hashing-Algorithmus im kryptographischen Bereich gilt.

Im nachfolgenden Beispiel wird ein Passwort „gehasht“. Diese Vorgehensweise sollte beispielsweise verwendet werden, um Passwörter in einer Datenbank zu speichern, welches z.B. im Rahmen des Softwareprojektes „Yachthafen Resort“ eine Anforderung ist. Es wird zum einen eine Variante in Python und zum anderen eine Variante in Java vorgestellt. Beide Varianten sollten aber für das gleiche Passwort auch den gleichen Hashwert liefern. Dass dieses der Fall ist, kann an den beiden Ausgaben gesehen werden.

SHA-256-Hashwert in Python und Java erzeugen

Um in Python einen SHA-256-Hashwert zu generieren, muss das Paket „hashlib“ importiert werden. Die Funktion „hashlib.sha256()“ wird verwendet, um ein SHA-256-Hashobjekt zu erstellen. „hexdigest()“ wandelt dann die kodierten Daten in ein hexadezimalen Format um. Umsetzung in Python

Umsetzung in Python

```
</> import hashlib
password = "Mein Passwort"
encoded_password = password.encode()
hash_value = hashlib.sha256(encoded_password)
print("Passwort: " + password)
print("Hashwert: " + hash_value.hexdigest())
```

Ausgabe

```
Passwort: Mein Passwort
Hashwert: 8b33bb3f2d7c098a6a07fb10ae4e3237ce0ab44df447906cbf7666b865b6b442
```

Java

Java bietet eine integrierte MessageDigest-Klasse für SHA-256-Hashing an. Diese muss mit dem Paket „java.security.MessageDigest“ in das Programm eingebunden werden. Danach wird ein Byte-Hashwert mit der Funktion „digest()“ erzeugt. Dieser wird im Anschluss durch einen Algorithmus in einen hexadezimalen Hashwert überführt.

Umsetzung in Java

```
</> import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;

public class Program {
    public static void main(String[] args) {
        try {
            String password = "Mein Passwort";
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] encodedPassword =
                digest.digest(password.getBytes(StandardCharsets.UTF_8));
            BigInteger no = new BigInteger(1, encodedPassword);
            String hashValue = no.toString(16);
            while (hashValue.length() < 32) {
                hashValue = "0" + hashValue;
            }
            System.out.println("Passwort: " + password);
            System.out.println("Hashwert: " + hashValue);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Byte-Hashwert in hexadezimalen Hashwert überführen

Ausgabe

```
Passwort: Mein Passwort
Hashwert: 8b33bb3f2d7c098a6a07fb10ae4e3237ce0ab44df447906cbf7666b865b6b442
```

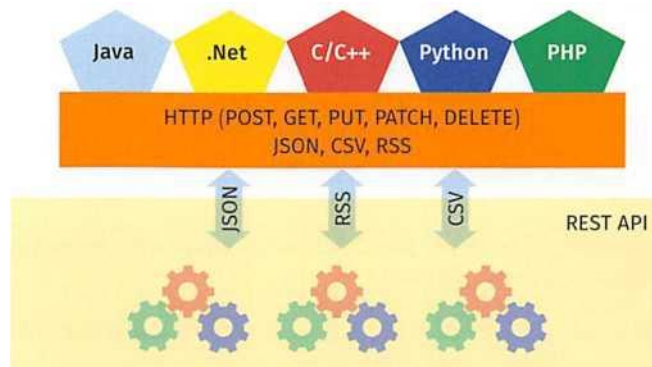
✓ Kompetenzcheck

1. Es soll der Hashalgorithmus von S. 183 so erweitert werden, dass der dezimale Hashwert in einen hexadezimalen Hashwert umgewandelt und als Zeichenkette zurückgegeben wird. Planen Sie zunächst den Algorithmus mithilfe eines Struktogramms und implementieren Sie anschließend die Lösung. (Hinweis: Bitte den Algorithmus zur Übung selbst entwickeln.)
2. Bearbeiten Sie im Arbeitsbuch die Lernsituation 3 mit den Aufgaben 8 und 9. Vertiefen Sie dabei mithilfe eines englischen Textes Ihr Wissen über Passwort-Hashing.

3.7. JSON-Dateien im- und exportieren und Webservices nutzen

- **Sie sollen für Softwarekomponenten den Im- und Export für Daten im JSON-Format planen und implementieren und Daten von angebotenen Webdiensten für Ihr Programm nutzen.**

Softwarekomponenten benötigen für ihre Arbeit die verschiedensten Daten. Diese Daten erhalten sie über Schnittstellen in Form von Eingabedaten. Dem EVA-Prinzip (siehe auch Jahrgangsband 1, Lernfeld 5) folgend werden die Eingangsdaten dann mithilfe von Algorithmen zu Ausgangsdaten verarbeitet. Diese Ausgangsdaten werden dann wieder über eine Schnittstelle zur weiteren Nutzung zur Verfügung gestellt.



Es gibt verschiedene Formen von Schnitt

stellen für eine Softwarekomponente, über die alle Daten ausgetauscht werden können. Das Lesen und Schreiben von Dateien gehört zu den grundlegendsten Formen für den Austausch von Daten. Die Daten in einer Datei liegen immer in einem bestimmten Format vor. Die Kenntnis des entsprechenden Dateiformats ist grundlegend für die Arbeit mit Dateien. Mit der Zeit haben sich Standardformate für verschiedene Bereiche herausgebildet, durch die es leichter ist, Daten zwischen unterschiedlichen Anwendungen auszutauschen. Besonders für den Austausch von Daten haben sich Standards wie CSV, XML oder JSON etabliert.

Exemplarisch für das Lesen und Schreiben von Dateien soll hier die Arbeit mit JSON-Dateien vorgestellt werden. Dieses Format spielt auch im Zusammenhang mit der Nutzung von Webdiensten eine große Rolle.

(1) JSON-Dateien lesen und schreiben

JSON ist ein Standardformat, das ähnlich wie XML zum Austausch von Daten zwischen Anwendungen dient und heutzutage sehr verbreitet ist. Der Aufbau einer JSON-Datei wurde bereits im zweiten Ausbildungsjahr behandelt. An dieser Stelle wird nun beispielhaft gezeigt, wie die Import- bzw. Exportfunktionalität von JSON-Dateien in Python umgesetzt werden kann. Hierzu werden Grundkenntnisse zum Öffnen und Schließen von Dateien in Python vorausgesetzt.

Um mit JSON-Daten zu arbeiten, verfügt Python über ein integriertes Paket namens „json“. Dieses muss in das Programm importiert werden. Mithilfe der Funktion „json.dumps“ können verschiedene Datentypen (Text, Zahl) und -Strukturen (Dictionary, Liste) in ein JSON-Objekt umgewandelt werden, welches dann z. B. ausgegeben oder in einer Datei gespeichert wird.

Eine ausführliche Dokumentation des Pakets „json“ ist unter <https://docs.python.org/3.10/library/json.html?highlight=json#module-json> zu finden.

Im nachfolgenden Beispiel werden verschiedene Mietdaten eines Liegeplatzes des Yachthafen Resorts in ein JSON-Objekt umgewandelt, zur Kontrolle der Konsole ausgegeben und als JSON-Datei exportiert.

Export von Daten als JSON-Datei

```

import json

liegeplatz_mietdaten = {
    "liegeplatzmietdaten": [
        {
            "liegeplatz_nr": 1234,
            "mieter": "Frank Maier",
            "mietdatum": "20.05.2021"},
        {
            "liegeplatz_nr": 999,
            "mieter": "Simone Schulz",
            "mietdatum": "05.03.2019"},
    ]
}

json_object = json.dumps(liegeplatz_mietdaten, indent = 4)
print(json_object)

with open("beispiel.json", "w") as outfile:
    json.dump(liegeplatz_mietdaten, outfile, indent = 4)

```

Beispieldatenstruktur. Die Daten liegen normalerweise im Programm schon vor.

Umwandeln der Daten mithilfe der Funktion „json.dumps(...)“ in ein JSON-Objekt. Dieses wird zur Kontrolle in der Konsole ausgegeben. Der Übergabeparameter „indent“ bewirkt eine Formatierung der Daten. Ohne Parameter werden diese hintereinandergeschrieben.

Öffnen einer Datei und Schreiben der Daten. Der „indent“-Parameter wird hier normalerweise weggelassen.

Die Ausgabe in der Konsole bzw. in der Datei ist nachfolgend zu sehen.

The left screenshot shows a code editor with the following JSON structure:

```

{
  "liegeplatzmietdaten": [
    {
      "liegeplatz_nr": 1234,
      "mieter": "Frank Maier",
      "mietdatum": "20.05.2021"
    },
    {
      "liegeplatz_nr": 999,
      "mieter": "Simone Schulz",
      "mietdatum": "05.03.2019"
    }
  ]
}

```

The right screenshot shows a file editor titled 'beispiel.json - Editor' with the following JSON structure:

```

{
  "liegeplatzmietdaten": [
    {
      "liegeplatz_nr": 1234,
      "mieter": "Frank Maier",
      "mietdatum": "20.05.2021"
    },
    {
      "liegeplatz_nr": 999,
      "mieter": "Simone Schulz",
      "mietdatum": "05.03.2019"
    }
  ]
}

```

Genauso wie die Daten als JSON-Datei exportiert werden, können Daten auch von JSON-Dateien importiert werden. Im Beispiel werden die Daten aus der eben erzeugten JSON-Datei wieder eingelesen und zunächst alle Daten und danach Teile davon in der Konsole angezeigt. Wie die eingelesenen Daten von der Software Komponente schließlich verarbeitet werden, hängt von dem zu lösenden Problem ab.

Import von Daten als JSON-Datei

```

import json

f = open('beispiel.json', )
data = json.load(f)
for i in data['liegeplatzmietdaten']:
    print(i)

number = data['liegeplatzmietdaten'][0]['liegeplatz_nr']
tenant = data['liegeplatzmietdaten'][0]['mieter']
print("Liegeplatz: " + str(number))
print("Mieter: " + tenant)
f.close()

```

Beim Ausführen des Programms erscheint folgende Konsolenausgabe:

```
{'liegeplatz_nr': 1234, 'mieter': 'Frank Maier', 'mietdatum': '20.05.2021'}
{'liegeplatz_nr': 999, 'mieter': 'Simone Schulz', 'mietdatum': '05.03.2019'}
Liegeplatz: 1234
Mieter: Frank Maier

Process finished with exit code 0
```

Ähnlich wie in Python können auch in Java JSON-Dateien gelesen und geschrieben werden. Dafür gibt es mehrere Möglichkeiten. Eine Variante ist, die Funktionalität des Pakets „javax.json“ zu nutzen. Allerdings gehört das Paket nicht zum Standardumfang des Software Development Kit (SDK), sondern es muss heruntergeladen und in das Projekt eingebunden werden. Eine Dokumentation dieser Schnittstelle ist bei Oracle zu finden.

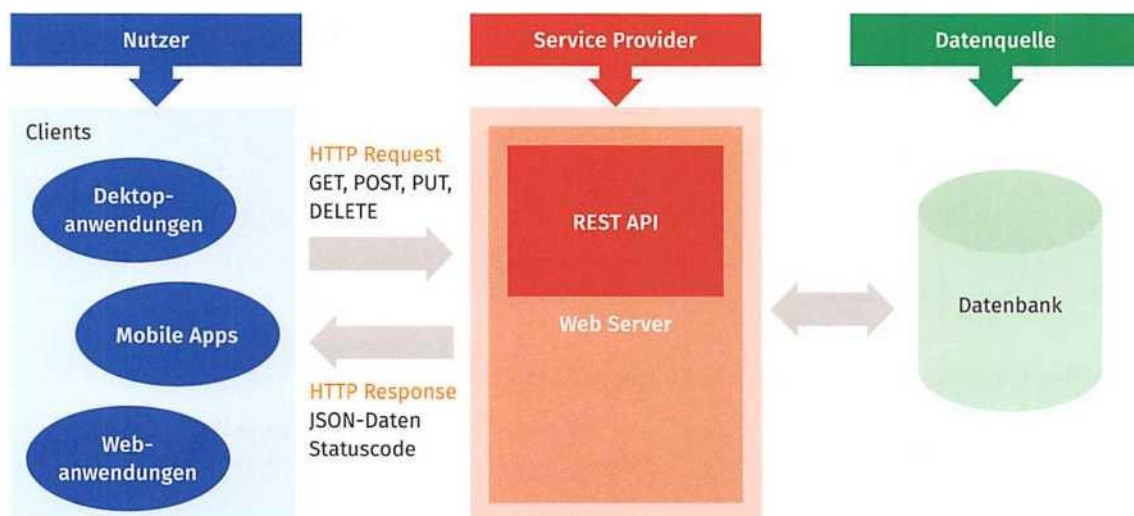
Interface	Description
JsonArray	JsonArray represents an immutable JSON array (an ordered collection of JSON values).
JsonArrayBuilder	A builder for creating JsonArray models from scratch.
JsonBuilderFactory	Factory to create JsonObjectBuilder and JsonArrayBuilder.
JsonNumber	An immutable JSON number value.

Download javax.json: <https://repo1.maven.org/maven2/javax/json/javax.json-api/1.1.4/>

Dokumentation von javax.json: <https://docs.oracle.com/javaee/7/api/index.html>

(2) Webservices nutzen

Eine Art von Datenquellen, welche von Programmen genutzt werden können, sind Datenquellen, welche über Webdienste im Internet zur Verfügung gestellt werden. Viele Webdienste, z. B. „YouTube“ oder „Git-Hub“, machen ihre Daten über eine API für Drittanbieter zugänglich. Eine wichtige API in diesem Zusammenhang ist die REST-API. Diese ermöglichen den Zugriff auf Webservices über öffentliche Web-URLs. Die komplette Architektur einer REST-Anwendung entspricht einer Client-Server-Architektur.



Die URL für die REST-API von „Google Maps“ ist beispielsweise <https://www.google.com/maps/embed/>.

Clients können auf die Daten zugreifen, indem sie eine HTTP-Anforderung an einen Webserver senden. Dieser gibt dann entsprechende Daten im JSON-Format zurück. Eine HTTP-Anforderung teilt der REST-API mit, welche Aktion ausgeführt werden soll. Dafür gibt es verschiedene HTTP-Methoden. Nachfolgend sind die vier wichtigsten Methoden aufgeführt.

Wichtige HTTP-Methoden	
GET	Die Methode dient zum Lesen und Abrufen von Daten.
POST	Die Methode dient zum Erstellen oder Aktualisieren einer Ressource.
PUT	Die Methode dient zum Erstellen oder Aktualisieren einer Ressource.
DELETE	Die Methode dient zum Löschen einer Ressource vom Server.

Der Unterschied zwischen POST und PUT besteht darin, dass das mehrmalige Aufrufen derselben PUT- Anforderung immer zum gleichen Ergebnis führt, während das wiederholte Aufrufen derselben POST- Anforderung dieselbe Ressource mehrmals erstellt. Sobald eine REST-API eine HTTP-Anforderung von einem Client empfängt, verarbeitet sie diese und gibt danach eine HTTP-Antwort zurück. In dieser Antwort ist auch ein HTTP-Statuscode enthalten. Dieser Code stellt Informationen zu den Ergebnissen der Anforderung bereit. Eine Anwendung kann den Statuscode überprüfen und gegebenenfalls entsprechende Aktionen ausführen, z. B. die Behandlung von Fehlern oder das Anzeigen einer Erfolgsmeldung.

HTTP-Statuscodes mit Beispielen	
Code	Kategorie
1XX	Informativ: Kommuniziert Informationen auf Übertragungsprotokollebene. 101: Wird als Antwort auf einen Upgrade-Anforderungsheader vom Client gesendet und gibt das Protokoll an, zu dem der Server wechselt.
2XX	Erfolg: Die Anfrage wurde erfolgreich bearbeitet. 200: OK - Die angeforderte Aktion war erfolgreich. 201: Eine neue Ressource wurde erstellt. 202: Angenommen - Die Anfrage ging ein, aber es wurde noch keine Änderung vorgenommen.
3XX	Umleitung: Es müssen zusätzliche Maßnahmen ergriffen werden, um die Anforderung zu beenden. 300: Die Anforderung hat mehr als eine mögliche Antwort. Es muss eine ausgewählt werden. 305: Gibt an, dass ein Proxy auf eine angeforderte Antwort zugreifen muss.
4 XX	Client-Fehler: Clientseitiger Fehler 400: Ungültige Anfrage 404: Die angeforderte Ressource wurde nicht gefunden. 415: nicht unterstützter Medientyp
5XX	Serverfehler: serverseitiger Fehler 500: Der Server hat bei der Verarbeitung der Anforderung einen Fehler ausgelöst. 501: Die HTTP-Methode wird vom Server nicht unterstützt und kann nicht verarbeitet werden.

Exemplarisch soll hier eine GET-Anforderung in Python implementiert werden. Python bietet verschiedene Werkzeuge an, um Daten von REST-API abzurufen oder auch eigene Python-REST-API zu erstellen. Durch den Import des Pakets „requests“ können u.a. nachfolgende Funktionen genutzt werden.

Python-Funktionen für REST-Anforderungen	
<code>get(url, params, args)</code>	sendet eine GET-Anforderung an die angegebene URL
<code>post(url, data, json, args)</code>	sendet eine POST-Anforderung an die angegebene URL
<code>put(url, data, args)</code>	sendet eine PUT-Anforderung an die angegebene URL
<code>delete(url, args)</code>	sendet eine DELETE-Anforderung an die angegebene URL

Beispielprogramm

```

import requests
import json

API_key = "API_KEY"
base_url = "https://api.openweathermap.org/data/2.5/weather?"
cityname = input("Geben Sie einen Ort ein: ")
url = base_url + "appid=" + API_key + "&q=" + cityname
weather_data = requests.get(url).json()
json_object = json.dumps(weather_data, indent = 4)
print("Aktuelle Wetterdaten von " + cityname + ":")
print(json_object)

```

Hier den eigenen Schlüssel einfügen

Abfragen der Daten

Das aufgeführte Beispiel ruft freiverfügbare Wetterdaten von <https://openweathermap.org/> ab und gibt diese in der Konsole aus. Dazu wird ein API-Schlüssel benötigt. Diesen erhält man, wenn man einen kostenlosen Account erstellt. Aus dem API-Schlüssel und dem Ortsnamen wird die URL zusammengebaut und dann an diese eine GET-Anforderung gesendet. Die zurückgegebenen Daten werden dann angezeigt. Da die Liste der Daten zu lang ist, ist nebenstehen nur ein Teil der Ausgabe dargestellt. Die abgerufenen Daten können nun weiterverarbeitet werden. So können diese Daten beispielsweise für verschiedene Module des Softwareprojektes „Yachthafen Resort“ von Bedeutung sein.

```
Geben Sie einen Ort ein: Warnemünde
Aktuelle Wetterdaten von Warnemünde:
{
  "coord": {
    "lon": 12.084,
    "lat": 54.1767
  },
  "weather": [
    {
      "id": 802,
      "main": "Clouds",
      "description": "scattered clouds",
      "icon": "03n"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 279.05,
    "feels_like": 276.11,
    "temp_min": 278.18,
    "temp_max": 279.87,
    "pressure": 1034,
```

✓ Kompetenzcheck

1. Geben Sie an, welche Aussagen richtig sind.
 - a. Der Import und der Export von Daten mithilfe von Dateien sind mögliche Datenschnittstellen für Softwarekomponenten.
 - b. JSON ist ein Format, um Daten z. B. in Dateien zu speichern.
 - c. EXECUTE ist eine HTTP-Methode, mit der REST-API angesprochen werden können.
 - d. Die HTTP-Methode „GET“ dient zum Lesen und Abrufen von Daten.
 - e. Die HTTP-Methode „DELETE“ dient zum Löschen einer Ressource vom Server.
 - f. Jeder HTTP-Statuscode, welcher mit einer 3 beginnt, z.B. 301, bestätigt auf irgendeine Art und Weise, dass die Anforderung erfolgreich bearbeitet wurde.
 - g. Jeder HTTP-Statuscode, welcher mit einer 4 beginnt, z. B. 400, bestätigt einen serverseitigen Fehler.
2. Bearbeiten Sie im Arbeitsbuch die Lernsituation 3 mit der Aufgabe 10.
3. Überprüfen Sie Ihr Wissen im Bereich Datenübertragungsformate und Webservices in Vorbereitung auf die Prüfung mithilfe der Aufgabe 11 der Lernsituation 3.

3.8. Funktionalität im Zusammenhang mit Datenbanken realisieren

➤ Sie sollen Verbindungen zu Datenbanken herstellen und SQL-Anweisungen ausführen.

Eine weitere Art von Datenquellen, welche von Programmen genutzt werden können, sind Datenbanken (siehe auch Jahrgangsband 1, Lernfeld 5, und Jahrgangsband 2, Lernfeld 8). Dazu müssen die Datenbanken über eine Schnittstelle angesprochen werden. Jede Datenbank stellt zunächst einmal eine eigene Schnittstelle bereit, über die andere Programme mit der Datenbank kommunizieren können. Über diese Datenbankschnittstelle wird die Herstellung einer Verbindung zur Datenbank ermöglicht und es wird der Datenaustausch zwischen der Anwendung und der Datenbank geregelt. Da die individuellen Schnittstellen der Datenbanken teilweise sehr unterschiedlich sind, muss theoretisch für jede Datenbank eine individuelle Lösung in der eigenen Softwarelösung implementiert werden. Dieses kann aber vermieden werden, wenn statt der herstellerspezifischen Schnittstelle eine allgemeine Datenbankschnittstelle, z. B. ODBC, genutzt wird.



Wichtige Datenbankschnittstellen

ODBC

ODBC (Open Database Connectivity) ist eine standardisierte Datenbank-API für den Zugriff auf Datenbankmanagementsysteme. Die ODBC-Schnittstelle ist unabhängig vom Datenbankmanagementsystem. ODBC erreicht diese Unabhängigkeit von Datenbankmanagementsystemen durch die Verwendung eines ODBC-Treibers, welcher zwischen der Anwendung und dem Datenbankmanagementsystem vermittelt. Die Anwendung verwendet die bereitgestellten ODBC-Funktionen und ODBC übergibt die Abfrage an das Datenbankmanagementsystem.

JDBC

JDBC (Java Database Connectivity) ist eine standardisierte Datenbank-API für Java-Plattformen, welche den Zugriff auf verschiedene Datenbankmanagementsysteme erlaubt. Es stellt Methoden zum Abfragen und Aktualisieren von Daten in einer Datenbank bereit und ist auf relationale Datenbanken ausgerichtet. Die Ergebnisse von SQL-Abfragen werden in eine Form umgewandelt, welche für Java nutzbar ist. Eine JDBC-zu-ODBC-Bridge ermöglicht Verbindungen zu jeder ODBC-zugänglichen Datenquelle. Für jedes Datenbankmanagementsystem sind eigene Treiber erforderlich, die die JDBC-Spezifikation implementieren. Diese werden i.d.R. vom Hersteller des Datenbankmanagementsystems bereitgestellt.

ADO.net

ADO.net ist die Datenzugriffstechnologie des .Net-Framework von Microsoft, welche die Kommunikation zwischen verschiedenen Datenbankmanagementsystemen und einem definierten Satz von Komponenten ermöglicht. Mithilfe dieser Komponenten kann auf die Daten einer Datenbank zugegriffen werden. ADO.net wird auch als Weiterentwicklung der ActiveX Data Objects (ADO)-Technologie angesehen, wurde aber so stark verändert, dass sie als völlig neues Produkt betrachtet werden kann.

ADO

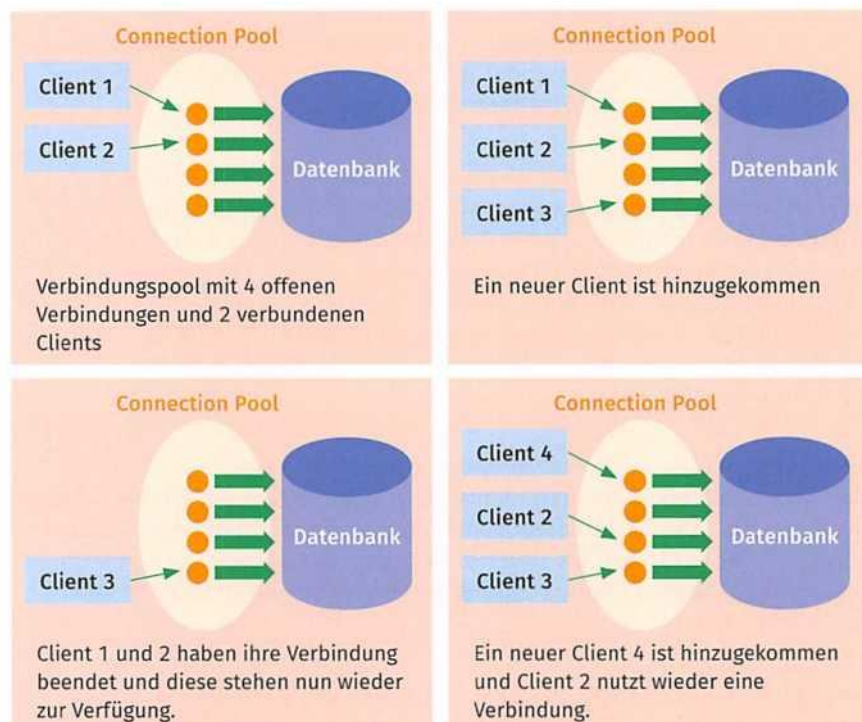
ADO (ActiveX Data Objects) ist eine auf ActiveX basierende Schnittstelle zum Datenzugriff auf Datenbanken und anderen Datenquellen. Für den Zugriff auf eine Datenbank bei Verwendung von ADO sind keine SQL-Kenntnisse erforderlich. Trotzdem kann ADO auch zum direkten Ausführen von SQL-Befehlen verwendet werden.

(1) Datenbankverbindung herstellen

Um mit einer Datenbank arbeiten zu können, muss zunächst erst einmal eine Verbindung über die Datenbankschnittstelle mit der Datenbank hergestellt werden. Verbindungen werden hergestellt, indem man der Datenbankschnittstelle verschiedene Informationen in Form von Schlüssel-Wert-Paaren, z. B. Anmeldeinformationen für die Benutzerauthentifizierung, zum Verbindungsaufbau zur Verfügung stellt.

Die notwendigen Informationen sind von der Datenbankschnittstelle genau vorgegeben. Sobald eine Verbindung hergestellt wurde, kann diese nach Belieben geöffnet und geschlossen werden. Zudem können Eigenschaften wie die Timeoutlänge eines Befehls oder einer Transaktion festgelegt werden. Um Aktionen auf der Datenbank auszuführen, muss diese zunächst geöffnet werden. Nach der Ausführung des Befehls wird die Datenbank wieder geschlossen.

Bei einfachen Operationen in kleinem Maßstab fallen die Schritte zum Öffnen und Schließen einer Verbindung nicht so ins Gewicht. Wenn die Anwendung jedoch nach oben skalieren soll, dann können sich das ständige Öffnen und Schließen von Verbindungen negativ auf die Leistung der Anwendung auswirken. Wird etwa nur eine einzelne Benutzertransaktion über eine Verbindung ausgeführt, dann ist der Verwaltungsaufwand für die Erstellung und Beendigung der Verbindung deutlich höher als die eigentlich geleistete Arbeit. Deswegen wurde das Verbindungspooling erfunden, um so die Leistung zu verbessern. Das Verbindungspooling ist eine Möglichkeit, die Kosten für das Öffnen und Schließen von Verbindungen zu senken, indem ein „Pool“ offener Verbindungen verwaltet wird, die bei Bedarf von einer Datenbankaktion an eine andere Datenbankaktion übergeben werden können. Das Verbindungspooling kann zur Verbesserung der Anwendungsleistung beitragen, ist jedoch nicht immer geeignet, da auch die Aufrechterhaltung einer offenen Verbindung mit Ressourcenkosten verbunden ist. Es hängt also immer von der spezifischen Aufgabenstellung ab, ob ein Verbindungspooling genutzt werden sollte oder nicht.



Nachfolgend sind beispielhaft das Erstellen und das Schließen einer Datenbankverbindung mit einer MySQL-Datenbank mithilfe von JDBC in Java dargestellt. Dabei müssen an den angezeigten Stellen der Benutzername und das Passwort durch konkrete Werte ersetzt werden.

Umsetzung in Java

```

import java.sql.*;

public class Program {
    public static void main(String[] args) {
        final String JDBC_DRIVER = "com.mysql.jdbc.Driver";

        final String URL = "jdbc:mysql://localhost/";
        final String USER = "benutzername";
        final String PASSWORD = "passwort";
        Connection con = null;
        try{
            Class.forName(JDBC_DRIVER);
            System.out.println("Datenbankverbindung wird hergestellt ...");
            con = DriverManager.getConnection(URL, USER, PASSWORD);
            System.out.println("Datenbankverbindung erfolgreich hergestellt!");
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (con != null) { con.close(); }
                System.out.println("Datenbank wurde geschlossen!");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

Hier eigenen Benutzernamen und Passwort angeben

JDBC-Treiber registrieren

Damit eine JDBC-Verbindung hergestellt werden kann, müssen zunächst die erforderlichen Pakete mit der Anweisung „import java.sql.*“ importiert und danach noch der richtige Treiber für die Datenbank angemeldet werden. In diesem Fall der JDBC-Treiber für MySQL-Datenbanken. Mit der Methode „getConnectionen(...)“ vom DriverManager wird die Verbindung zum Datenbankmanagementsystem hergestellt. Dabei werden als Parameter die URL, der Benutzername und das Passwort übergeben. Konnte die Verbindung erfolgreich hergestellt werden, können nun SQL-Anweisungen mithilfe von ausgeführt werden (hierauf

```

@ Javadoc Declaration Console X
<terminated> Program (4) [Java Application] C:\Users\Lokal
Datenbankverbindung wird hergestellt ...
Datenbankverbindung erfolgreich hergestellt!
Datenbank wurde geschlossen!

```

```

Statement st = con.createStatement();
String sql = "...";
st.executeUpdate(query); oder st.executeQuery(sql);

```

wurde im Beispiel verzichtet). Am Ende muss die Verbindung wieder mit der Methode „close()“ geschlossen werden.

Auch das Erstellen einer Datenbankverbindung mithilfe von Python ist ähnlich einfach. Python stellt dafür verschiedene Schnittstellen zur Verfügung, mit deren Hilfe z.B. MySQL-, SQLite-, MS SQL- oder PostgreSQL-Datenbanken angesprochen werden können. Dazu müssen die entsprechenden Treiber auf dem System installiert und die Pakete dann in Python zur Nutzung importiert werden.

(2) SQL zur Interaktion mit relationalen Datenbanken nutzen

Wenn eine Datenbankverbindung hergestellt und die Datenbank geöffnet wurden, dann können Aktionen auf der Datenbank ausgeführt werden. Bei relationalen Datenbanken wird dazu SQL verwendet (siehe Jahrgangsband 1, Lernfeld 5 und Jahrgangsband 2, Lernfeld 8).

Ein Überblick über die wichtigsten SQL-Befehle ist in der folgenden Tabelle zusammengestellt.

Übersicht SQL-Befehle	
Datenbanken, Nutzer	
CREATE DATABASE <i>datenbankname</i>	leere Datenbank anlegen
DROP DATABASE <i>datenbankname</i>	Datenbank löschen
CREATE USER <i>name</i> IDENTIFIED BY <i>pass-word</i>	Nutzer anlegen
Übersicht SQL-Befehle	
DROP USER <i>name</i>	Nutzer löschen
GRANT <i>rechte</i> ON <i>objekt</i> TO <i>nutzer</i>	Rechte vergeben
REVOKE <i>rechte</i> ON <i>objekt</i> FROM <i>nutzer</i>	Rechte entziehen
Tabellen	
CREATE TABLE <i>tabellenname</i> (<i>spaltenname</i> <i>datentyp</i> , ... PRIMARY KEY (<i>spaltenname</i>), FOREIGN KEY (<i>spaltenname</i>) REFERENCES <i>tabellenname</i> (<i>spaltenname</i>))	leere Tabelle anlegen Datentypen: INTEGER, DATE, DOUBLE, FLOAT, CHARACTER(<i>n</i>), VARCHAR(<i>n</i>), BOOLEAN
ALTER TABLE <i>tabellenname</i> ADD COLUMN <i>spaltenname</i> <i>datentyp</i> MODIFY COLUMN <i>spaltenname</i> <i>datentyp</i> DROP COLUMN <i>spaltenname</i>	Tabelle ändern: Spalte hinzufügen Datentyp einer Spalte ändern Spalte löschen
DROP TABLE <i>tabellenname</i>	Tabelle löschen
CREATE INDEX <i>indexname</i> ON <i>tabellenname</i> (<i>spaltenname</i>)	Index erstellen
DROP INDEX <i>indexname</i> ON <i>tabellenname</i>	Index löschen
Operatoren	
AND, OR, NOT	logische Operatoren
=, <=, >=, <, >, <>	Vergleichsoperatoren
=	Zuweisungsoperator
+, -, /, *	Rechenoperatoren
LIKE Platzhalter: % _	Überprüfung von Text auf ein bestimmtes Muster
IS NULL, IS NOT NULL	Überprüfung auf NULL bzw. auf nicht NULL
Datensätze	
INSERT INTO <i>tabelle</i> (<i>spalte1</i> , ...) VALUES (<i>wert1</i> , ...)	Datensatz einfügen
UPDATE <i>tabelle</i> SET <i>spalte</i> = <i>wert</i> WHERE <i>bedingungen</i>	Datenwerte aktualisieren
DELETE FROM <i>tabelle</i> WHERE <i>bedingungen</i>	Datensatz löschen
Abfragen	
SELECT <i>spalte1</i> , ... FROM <i>tabelle 1</i> , ... WHERE <i>bedingungen</i>	Daten abfragen
DISTINCT	Redundanzen aus der Ergebnismenge entfernen
ORDER BY <i>spalte1</i> , ... ASC DESC	Ergebnismenge sortieren (aufsteigend absteigend)

Übersicht SQL-Befehle	
GROUP BY <i>spalte</i> HAVING <i>bedingungen</i>	Ergebnismenge gruppieren
<i>tabelle1</i> INNER JOIN <i>tabelle2</i> ON <i>tabelle1.pk</i> = <i>tabelle2.fk</i>	Ergebnismenge enthält alle Werte der beiden Tabellen, welche miteinander verknüpft sind
<i>tabelle1</i> LEFT JOIN <i>tabelle2</i> ON <i>tabelle1.pk</i> = <i>tabelle2.fk</i>	Ergebnismenge enthält alle Werte der linken Tabelle und alle Werte der rechten Tabelle, welche mit der linken Tabelle verknüpft sind
<i>tabelle1</i> RIGHT JOIN <i>tabelle2</i> ON <i>tabelle1.pk</i> = <i>tabelle2.fk</i>	Ergebnismenge enthält alle Werte der rechten Tabelle und alle Werte der linken Tabelle, welche mit der rechten Tabelle verknüpft sind
<i>tabelle1</i> FULL JOIN <i>tabelle2</i> ON <i>tabelle1.pk</i> = <i>tabelle2.fk</i>	Ergebnismenge enthält alle Werte der linken Tabelle und rechten Tabelle
SELECT <i>spalte</i> FROM <i>tabelle1</i> UNION SELECT <i>spalte</i> FROM <i>tabelle2</i>	Ergebnismengen kombinieren
SELECT ... (SELECT ... FROM ...) FROM ...	Unterabfragen Mengenoperatoren: IN , EXISTS , ANY , ALL
Aggregatfunktionen	
MIN (<i>spalte</i> oder <i>formel</i>)	kleinsten Wert ermitteln
MAX (<i>spalte</i> oder <i>formel</i>)	größten Wert ermitteln
AVG (<i>spalte</i> oder <i>formel</i>)	Durchschnitt aller Werte ermitteln
SUM (<i>spalte</i> oder <i>formel</i>)	Summe aller Werte ermitteln
COUNT (<i>spalte</i> oder *)	Anzahl aller Nicht-NULL-Werte ermitteln

SQL-Statements können nun durch verschiedene Funktionen der Datenbank zur Ausführung übermittelt werden. Dazu werden die SQL-Anweisungen der entsprechenden Funktion als String übergeben.

Funktionen, um in Java SQL-Anweisung auszuführen

```

</> public void updateDB(String sql) {
    try {
        sqlStatement.executeUpdate(sql);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public ResultSet getData(String sql) {
    try {
        return sqlStatement.executeQuery(sql);
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

```

Mit „executeUpdate(...)“ können u.a. folgende Anweisungen ausgeführt werden:
INSERT INTO, UPDATE, DELETE, CREATE TABLE, ALTER TABLE, DROP TABLE

Mithilfe von „executeQuery(...)“ kann einer SELECT-Anweisung ausgeführt werden. Die Ergebnismenge wird als ResultSet an das Programm zurückgegeben.

Weiterhin sind bei der Implementierung der Funktionalität im Zusammenhang mit relationalen Datenbanken und SQL auch Transaktionen, Stored Procedures und Trigger von Bedeutung.

(3) Transaktionen

Im Zusammenhang mit Datenbanken ist eine Transaktion eine Folge von Datenbankmanipulationen, die entweder vollständig oder gar nicht abgeschlossen werden und die die Datenbank in einem konsistenten Zustand belassen. Transaktionen sind z. B. von Bedeutung, wenn mehrere Benutzer gleichzeitig auf dieselben Datenbestände zugreifen und diese manipulieren wollen. Dabei können sich die Änderungen gegenseitig beeinflussen und zu inkonsistenten

Datenbeständen führen. Die Konsistenz von Daten hat aber bei relationalen Datenbanken höchste Priorität. Deswegen werden die Datensätze während ihrer Manipulation für konkurrierende Zugriffe gesperrt.

Das Transaktionssystem zielt darauf ab, möglichst viele Transaktionen in möglichst kurzer Zeit zu bedienen. Eine serielle Realisierung der Transaktionen ist zwar einfach, liefert aber eine schlechte Performance. Aus diesem Grund werden Transaktionen in einzelnen Operationen aufgeteilt und ausgeführt. Diese können erfolgreich ausgeführt werden oder fehlschlagen. Im ersten Fall (Commit) wurde die Transaktion problemlos beendet und das Ergebnis dauerhaft abgespeichert. Im zweiten Fall (Abort) sind Probleme aufgetreten und die Ausführung wurde nicht fortgesetzt. In diesem Fall müssen durch ein Rollback sämtliche Auswirkungen der Transaktion auf den Datenbestand rückgängig gemacht und die Datenbank in ihren Ausgangszustand zurückversetzt werden. Für diesen Vorgang kennt SQL folgende Befehle:

SQL-Befehle im Zusammenhang mit Transaktionen	
Befehl	Beschreibung
START TRANSACTION oder BEGIN	startet eine Transaktion
COMMIT	mit dieser Anweisung werden die Änderungen infolge von Datenbankmanipulationen dauerhaft in der Datenbank gespeichert
ROLLBACK	mit dieser Anweisung werden die Änderungen infolge von Datenbankmanipulationen rückgängig gemacht, d. h., es wird der Zustand vor der Transaktion hergestellt

Im ersten Beispiel sollen in einer SQL-Transaktion mit COMMIT zwei SQL-Befehle ausgeführt und die Änderungen dauerhaft in die Datenbank übernommen werden.

```
</> START TRANSACTION;
INSERT INTO Artikel (Name, Preis) VALUES ("Ansichtskarte", 1.25);
DELETE FROM Artikel WHERE Artikel.ID = 712;
COMMIT;
```

Im zweiten Beispiel wird im Rahmen einer SQL-Transaktion mit ROLLBACK zu Testzwecken der Preis aller Artikel gelöscht und die geänderten Daten zur Kontrolle abgerufen. Anschließend werden die Änderungen wieder rückgängig gemacht.

```
</> BEGIN;
UPDATE Artikel SET Preis = NULL;
SELECT Artikel.* FROM Artikel;
ROLLBACK;
```

Auch JDBC bietet die Möglichkeit, Transaktionen auszuführen. Das Standardverhalten einer Datenbankverbindung mit JDBC ist Auto-Commit, d. h., dass jede einzelne Anweisung als Transaktion behandelt und direkt nach der Ausführung automatisch festgeschrieben wird. Es können aber auch mehrere SQL-Anweisungen in einer Transaktion gebündelt werden. Dieses kann wie in dem Beispiel aufgeführt geschehen:

```
</> Connection con = DriverManager.getConnection(CONNECTION_URL, USER,
                                                PASSWORD);

try {
    con.setAutoCommit(false); // Standardverhalten von JDBC ausschalten
    PreparedStatement ps1 =
        con.prepareStatement("INSERT INTO Artikel(Name, Preis)
                              VALUES ('Ansichtskarte', 1.25)");
    ps1.executeUpdate();
    PreparedStatement ps2 =
        con.prepareStatement("DELETE FROM Artikel
                              WHERE Artikel.ID = 712");
    ps2.executeUpdate();
    con.commit(); // Änderungen dauerhaft in der Datenbank speichern
} catch (Exception e) {
    con.rollback(); // Beim Fehler die komplette Datenbanktransaktion rückgängig machen
}
```

(4) Stored Procedures und Trigger

Stored Procedures sind gespeicherte Prozeduren (Unterprogramme), mit deren Hilfe ganze Abläufe von Anweisungen ausgeführt werden können. Sie stellen eigenständige Befehle dar, welche wiederum eine Abfolge gespeicherter Befehle ausführen. Neben SQL-Anweisungen können Stored Procedures auch zusätzliche Befehle zur Ablaufsteuerung und Auswertung von Bedingungen enthalten. Die Vorteile von Stored Procedures liegen in einer höheren Verarbeitungsgeschwindigkeit und einer besseren Systemsicherheit, was den Anforderungen der Informationssicherheit entspricht.

Das Erstellen und Verwenden von Stored Procedures ist ein umfangreiches Thema. An dieser Stelle wird nur ein Einblick in dieses Thema gegeben. Auch hier gilt, dass sich die Syntax von Datenbankmanagementsystem zu Datenbankmanagementsystem unterscheiden kann.

Erstellen einer Stored Procedures	
Syntax	<code>CREATE PROCEDURE Prozedurname AS Anweisungsblock;</code>
Erläuterung	Prozedurname = Name der Prozedur für die interne Verwaltung Anweisungsblock = enthält SQL-Anweisungen und Befehle zur Ablaufsteuerung
Beispiel	<code>CREATE PROCEDURE ErmittleOrte @Landname VARCHAR(50), AS SELECT Land.Name, Ort.Name FROM Ort INNER JOIN Land ON Ort.LandID = Land.LandID WHERE Land.Name = @Landname;</code>
	Ausführen der Stored Procedure:
	<code>EXECUTE ErmittleOrte('Deutschland');</code>

Löschen einer Stored Procedures	
Syntax	<code>DROP PROCEDURE Prozedurname;</code>
Erläuterung	Prozedurname = Name der Prozedur, die gelöscht werden soll
Beispiel	<code>DROP PROCEDURE ErmittleOrte;</code>

Datenbanktrigger

Datenbanktrigger (oder kurz: Trigger) stellen eine besondere Form der Stored Procedures dar, welche nicht von dem Anwender, sondern von der Datenbank selbst gestartet werden.

✓ Kompetenzcheck

- Geben Sie an, welche Aussagen richtig sind.
 - ODBC ist eine standardisierte Datenbankschnittstelle.
 - Ein Connection Pool ist eine Sammlung von Datenbankverbindungen, welche aus Effizienzgründen ständig offengehalten werden.
 - Eine Transaktion ist eine SQL-Anweisung, welche mindestens zwei Unterabfragen enthält.
 - Mit dem Schlüsselwort BEGIN wird eine Transaktion gestartet.
 - COMMIT und ROLLBACK werden im Zusammenhang mit Transaktionen verwendet.
 - Unter JDBC wird standardmäßig jede SQL-Anweisung als einzelne Transaktion betrachtet.
 - Stored Procedures sind eine spezielle Form der Datenbanktrigger.
 - Stored Procedures können neben den SQL-Anweisungen u.a. auch Verzweigungen enthalten.
- Überprüfen Sie als Prüfungsvorbereitung Ihr Wissen zu Datenbanken und SQL, indem Sie die Aufgaben 12 und 13 der Lernsituation 3 im Arbeitsbuch bearbeiten.

4. Funktionalität von Softwarekomponenten testen

- **Sie sollen die Funktionalität von Softwarekomponenten testen. Dazu bestimmen Sie die erforderlichen Testfälle und implementieren Unittests.**

Das Testen als solches ist Bestandteil des Qualitätssicherungsprozesses in der Softwareentwicklung und betrifft sowohl die funktionalen als auch die nicht funktionalen Qualitätsmerkmale einer Software. In diesem Kapitel soll das Testen der Funktionalität von Softwarekomponenten bzw. -modulen im Vordergrund stehen. Dieses erfolgt in der Regel durch sogenannte Modultests (Komponententest, Unittest). Modultests stellen die unterste Ebene in der Teststufenhierarchie dar, unterscheiden sich aber - je nach Vorgehensmodell - durch ihren Verwendungszeitpunkt und ihre Rolle. Während sie in klassischen Vorgehensmodellen vornehmlich der Suche von funktionalen Fehlern dienen, ermöglicht diese Art von Tests in agilen Vorgehensmodellen erst eine effektive iterative Entwicklung. In diesem Zusammenhang gewinnt vor allem die Automatisierung des Testprozesses enorm an Bedeutung.



4.1. Testverfahren auswählen

- **Sie sollen ein geeignetes Testverfahren zum Testen Ihrer Softwarekomponente auswählen.**

Testverfahren lassen sich in statische und dynamische Testverfahren einteilen. Auch wenn mit statischen Testverfahren, z.B. einer Review, die Funktionalität einer Komponente überprüft werden kann, sind diese Verfahren für ein wiederholtes Testen von Softwarekomponenten zu aufwendig und werden in diesem Zusammenhang oftmals nur zur Überprüfung von speziellen Algorithmen angewandt. Für das Testen der funktionalen Anforderungen eignen sich dynamische Testverfahren besser, zumal diese auch automatisiert werden können.

Zu den dynamischen Testverfahren zählen der Black-Box- und der White-Box-Test. Beim White-Box-Test liegt der Quelltext vor und es wird versucht, eine möglichst hohe Testabdeckung bezüglich des Codes zu erreichen. Deswegen wird ein White-Box-Test auch als ein strukturiertes Testverfahren betrachtet. Der Black-Box-Test hingegen ist ein spezifikationsorientiertes Verfahren, weil hier nicht der Quellcode, sondern nur die Schnittstelle und deren Anforderungsspezifikationen getestet werden. Hier wird versucht, möglichst eine hohe Testabdeckung bezüglich der funktionalen Anforderungen zu erreichen.

Testabdeckung

Die Testabdeckung bezeichnet den Grad, mit dem ein zu testender Bereich durch Tests abgedeckt wird. Beispiele für solche Bereiche sind die Abdeckung des Codes, der Daten oder der funktionalen Anforderungen.

An dieser Stelle stellt sich die Frage, ob man White-Box-Test oder Black-Box-Test einsetzen sollte. Dazu sollten die Vor- und Nachteile betrachtet und für jeden einzelnen Fall gegeneinander abgewogen werden.

Black- und White-Box-Tests im Vergleich

Black-Box-Test	
Vorteile	Nachteile
Vorteile - nicht an den Quellcode gekoppelt, d. h., solange keine Änderungen an der Anforderungsspezifikation vorliegen, können sie immer wieder zum Testen der Komponente verwendet werden - Testfälle können sofort nach Abschluss der Spezifikation entworfen werden - es wird aus der Sicht des Anwenders getestet - sehr effizient für größere Komponenten und Module	Nachteile - Testfälle aus einer vollständigen Anforderungsspezifikation abzuleiten kann sehr aufwendig sein - aus unvollständigen und ungenauen Anforderungsspezifikationen können kaum alle notwendigen Testfälle abgeleitet werden, d. h., es wird keine hohe Testabdeckung erreicht - bietet keine Garantie dafür, dass das Programm vollständig getestet wird. Es kann immer Programmteile geben, die zusätzlich eingefügt wurden, aber nicht direkt durch die Anforderungsspezifikation erfasst werden. - es kann nur festgestellt werden, dass die Komponente nicht korrekt funktioniert; der fehlerauslösende Teil bleibt zunächst einmal unbekannt; dazu sind White-Box-Tests notwendig zum Testen komplexer Algorithmen ungeeignet

White-Box-Test	
Vorteile	Nachteile
• es können interne Fehler gefunden und lokalisiert werden • es werden ebenfalls alle Programmteile getestet, welche durch die Anforderungsspezifikation nicht erfasst wurden; i.d.R. wird eine hohe Testabdeckung des Quellcodes erreicht, wenn der Quelltext nicht zu komplex ist • komplexe Algorithmen können effektiv getestet werden • unterstützt das Optimieren des Quellcodes	• sind sehr stark an den Quellcode gekoppelt, d.h., sie verlieren nach Erweiterungen und Änderungen ihre Gültigkeit und müssen angepasst werden • bei komplexem Quellcode ist eine hohe Testabdeckung sehr aufwendig zu realisieren • können erst ihre endgültige Fassung erreichen, wenn der Code komplett fertiggestellt ist

In vielen Fällen stellt sich allerdings gar nicht die Entweder-oder-Frage, sondern es werden beide Formen nebeneinander eingesetzt. Außerdem hängt die Unterscheidung zwischen White- bzw. Black-Box-Test auch stark vom Kontext ab, in dem diese verwendet werden. Derselbe Test einer Komponente kann aus Sicht des Gesamtsystems ein White-Box-Test sein, aber bezogen auf die Komponente selbst eventuell ein Black-Box-Test.

Als nächsten Schritt gilt es zu entscheiden, ob die Tests manuell oder automatisiert durchgeführt werden sollen.

Manuelles und automatisiertes Testen im Vergleich	
Manuelles Testen	Automatisiertes Testen
Beim manuellen Testen wird die Software händisch ohne ein Automatisierungstool von einer bestimmten Person getestet. Der Tester sitzt tatsächlich vor dem Bildschirm, führt alle geplanten Tests durch und protokolliert die Ergebnisse. • Es ist immer eine Person zum Testen notwendig. • Diese Art des Testens erfordert viel Zeit und es sind teilweise erfahrene Tester notwendig. Dies ist daher mit hohen Kosten verbunden. • Es kann jede Art von Anwendung manuell getestet werden. • Die ständige Wiederholung von gleichen Tests kann die Tester schnell langweilen und ermüden.	Beim automatisierten Testen wird zunächst ein Programm geschrieben, mit dessen Hilfe weitere geplante Tests automatisch und unabhängig von einem Tester ausgeführt werden. Auch die Testergebnisse werden automatisch protokolliert. • Automatisiertes Testen spart Zeit, Kosten und menschliche Ressourcen. Einmal implementiert, ist es einfach, einen automatisierten Testlauf ständig zu wiederholen. • Automatisiertes Testen wird häufig für Regressionstests verwendet. • Automatisiertes Testen ist zuverlässiger, da es menschliche Fehler vermeidet. • Der Anfangsaufwand ist durch das Schreiben der Tests höher. • Nicht alle Arten von Anwendungen können automatisiert getestet werden. • Bei Änderungen der Anforderungsspezifikationen oder des Quellcodes müssen die Testprogramme jedes Mal angepasst werden. Hier kann es zu Fehlern kommen.

Aufgrund des rasanten Entwicklungstempos und der ständig steigenden Anforderungen im Bereich der Softwareentwicklung müssen auch die Testverfahren angepasst werden. Manuelle Tests sind in diesem Umfeld nicht ausreichend skalierbar. Hier bieten automatisierte Tests eine schnellere, genauere und vor allem eine gut skalierbare Alternative.

Regressionstests

Regressionstests sind Tests, mit deren Hilfe sichergestellt werden soll, dass das Hinzufügen von neuem Code, Optimierungen innerhalb des Codes oder das Beheben von Fehlern die vorhandene Funktionalität nicht beeinträchtigt und noch den Anforderungen entspricht.

Automatisiertes Testen bedeutet aber nicht, den gesamten manuellen Testprozess zu ersetzen, denn es gibt immer Bedingungen, unter denen automatisiertes Testen nicht eingesetzt werden kann. Auch kann der Aufwand zum Schreiben dieser Tests für das zu testende Objekt nicht lohnenswert sein. Des Weiteren kann, wenn die Software häufigen Veränderungen unterliegt, der Aufwand der Anpassung der Unittests schnell die zeitlichen Vorteile des automatisierten Testes wieder negieren und die Arbeit ausbremsen. Manchmal soll das Testen auch genutzt werden, um das Verhalten einer Software kennenzulernen und dieses zu studieren. Auch dieses Anliegen kann durch automatisiertes Testen nicht erreicht werden. Allerdings schließt das eine das andere nicht aus: So können große Teile einer Komponente beispielsweise mit automatisierten Unittests getestet werden und spezielle, sehr wichtige Bereiche werden zusätzlich einer manuellen Überprüfung unterzogen. Das Testen zu automatisieren bedeutet in der Regel, dass routinemäßige und sich wiederholende Testfälle automatisiert werden, um sich auf komplexere Testszenarien manuell

konzentrieren zu können.

✓ Kompetenzcheck

1. Geben Sie an, welche Aussagen richtig sind.
 - a. Beim Testen einer Software schließen sich Black -und White-Box-Tests gegenseitig aus.
 - b. Statische Prüfverfahren sind zum Testen der Funktionalität dynamischen Testverfahren vorzuziehen, weil statische Testverfahren besser automatisiert werden können.
 - c. Eine hohe Anzahl an Testfällen bewirkt in jedem Fall eine hohe Testabdeckung.
 - d. Die Grundlage für White-Box-Tests bildet die Anforderungsspezifikation.
 - e. Durch Black-Box-Tests wird sichergestellt, dass jeder Programmteil getestet wird.
 - f. Bei automatisierten Tests wird ein Programm geschrieben, welches nach dem Start die implementierten Tests automatisch durchführt.
 - g. Automatisierte Tests sind manuellen Test immer vorzuziehen.
 - h. Bei Programmänderungen kann es notwendig werden, auch die automatisierten Tests anzupassen.
2. Bearbeiten Sie im Arbeitsbuch die Lernsituation 4 mit der Aufgabe 1 und wählen Sie für verschiedene Szenarien Testverfahren aus.

4.2. Testfälle formulieren

- **Sie sollen zum Testen einer Softwarekomponente entsprechende Testfälle bestimmen und auswählen.**

Wie bereits festgestellt eignen sich sowohl Black- als auch White-Box-Tests zum Testen der Funktionalität einer Softwarekomponente. In beiden Fällen stellt sich aber die Frage, welche und wie viele Testdaten man zum Testen verwendet. Das Grundziel ist es zunächst, eine hohe Testabdeckung zu erreichen. Deshalb gibt es Verfahren, welche bei der Auswahl der Testdaten unterstützen.



Verfahren beim Black-Box-Test sind beispielsweise die Äquivalenzklassenbildung oder die Grenzwertanalyse (siehe auch Jahrgangsband 2, Lernfeld 8).

Grundlage für die Gewinnung entsprechender Testdaten ist beim Black-Box-Test die Anforderungsspezifikation. Etwas anders verhält es sich beim White-Box-Test. Hier wird das Programm auf der Grundlage der Anforderungsspezifikation erstellt. Allerdings werden die Testdaten dann vor allem aus der Programmlogik abgeleitet. Zwei der Verfahren, welche hierbei zum Einsatz kommen, sind in der nachfolgenden Tabelle aufgeführt.

Metrik	Beschreibung
Anweisungs- überdeckung	Bei der Anweisungsüberdeckung wird angestrebt, dass jede Anweisung im Quellcode durch das Testen mindestens einmal ausgeführt wird. Wird dieses Ziel erreicht, dann wurde eine vollständige Anweisungsüberdeckung (100%) erreicht. In diesem Fall ist sichergestellt, dass kein Code existiert, welcher nie aufgeführt wird („toter Code“). Vorteile: • es werden nicht ausführbare Codeteile ermittelt • ist schnell zu realisieren • Testaufwand wird minimiert Nachteile: • jede Anweisung im Quellcode hat die gleiche Gewichtung • leere Zweige werden nicht ermittelt • niedrige Fehlerentdeckungsquote Beispiel: <pre> ergebnis = 0 if (a < b): a = -a ergebnis = a + b print(str(ergebnis)) </pre> Bei richtiger Auswahl der Testdaten wird nur ein Testfall benötigt, um eine vollständige Anweisungsüberdeckung zu erreichen. Dieses ist immer dann der Fall, wenn der Testwert für a kleiner als für b ist, z. B. a = 10 und b = 20.
Zweig- überdeckung	Bei der Zweigüberdeckung (auch Entscheidungs- oder Kantenüberdeckung) wird das Verhältnis zwischen ausgeführten und möglichen Programmzweigen gemessen. Man muss jeden Zweig eines Programms mindestens einmal betreten, um eine einhundertprozentige Zweigüberdeckung zu erreichen. Eine einhundertprozentige Zweigüberdeckung schließt immer eine einhundertprozentige Anweisungsüberdeckung mit ein. Im Gegensatz zu der Anweisungsüberdeckung müssen allerdings auch leere Pfade ohne Anweisungen, z. B. Else-Zweige, einmal betreten werden. Vorteile: • es werden nicht erreichbare Programmzweige aufgedeckt • Fehlerentdeckungsrate höher als bei der Anweisungsüberdeckung Nachteile: • Abhängigkeiten zwischen Bedingungen werden nicht berücksichtigt • Schleifen können nur unzureichend getestet werden • komplexe Verzweigungsbedingungen werden unzureichend getestet Beispiel: <pre> ergebnis = 0 if (a < b): a = -a ergebnis = a + b print(str(ergebnis)) </pre> Im Gegensatz zur Anweisungsüberdeckung ist mehr als ein Testfall nötig, um alle Zweige zu durchlaufen. Zum einen werden Testdaten benötigt, mit denen die Verzweigung ausgeführt wird. Es können hier z. B. die Daten von der Anweisungsüberdeckung a = 10 und b = 20 verwendet werden. Zum anderen muss aber auch mit Daten getestet werden, bei denen die Verzweigung nicht durchlaufen wird, z. B. a = 15 und b = 5.

(1) Beispiel für die Ermittlung der Testdaten für einen Black-Box-Test

Als Beispiel sollen nun die Testdaten für einen Black-Box-Test für einen Algorithmus ermittelt werden. Dieser soll ermitteln, ob die Yachtdaten einer Yacht wie Länge, Breite und Tiefgang den Gegebenheiten des Yachthafens entsprechen. Grundlage dafür bildet die Anforderungsspezifikation, welche hier noch einmal kurz dargestellt wird.

Beispielaufgabe

Eingabedaten:

Länge, Breite und Tiefgang der Yacht in Metern (zwei Stellen nach dem Komma) Die Eingabedaten werden dem Algorithmus als Parameter übergeben und mit internen Werten verglichen. Dabei gelten folgende Regeln für geeignete Yachten: Länge: zwischen 0 und 20 m, Breite: zwischen 0 und 5 m, Tiefgang: zwischen 0 und 3 m

Ausgabedaten:

True - geeignet für den Yachthafen

False - nicht geeignet für den Yachthafen

In einem ersten Schritt werden nun die Basisäquivalenzklassen gebildet.

Äquivalenzklasse	Wertebereich
1. gültige Äquivalenzklasse (Klasse mit zulässigen Werten)	Alle Zahlenwerte, die folgende Bedingungen erfüllen: Länge: zwischen 0 und 20 m, Breite: zwischen 0 und 5 m, Tiefgang: zwischen 0 und 3 m
2. ungültige Äquivalenzklasse (Klasse mit unzulässigen Werten)	Annahme: Es werden nur Zahlen als Parameter übergeben; alle Zahlenwerte, die nicht in die gültige Äquivalenzklasse fallen.

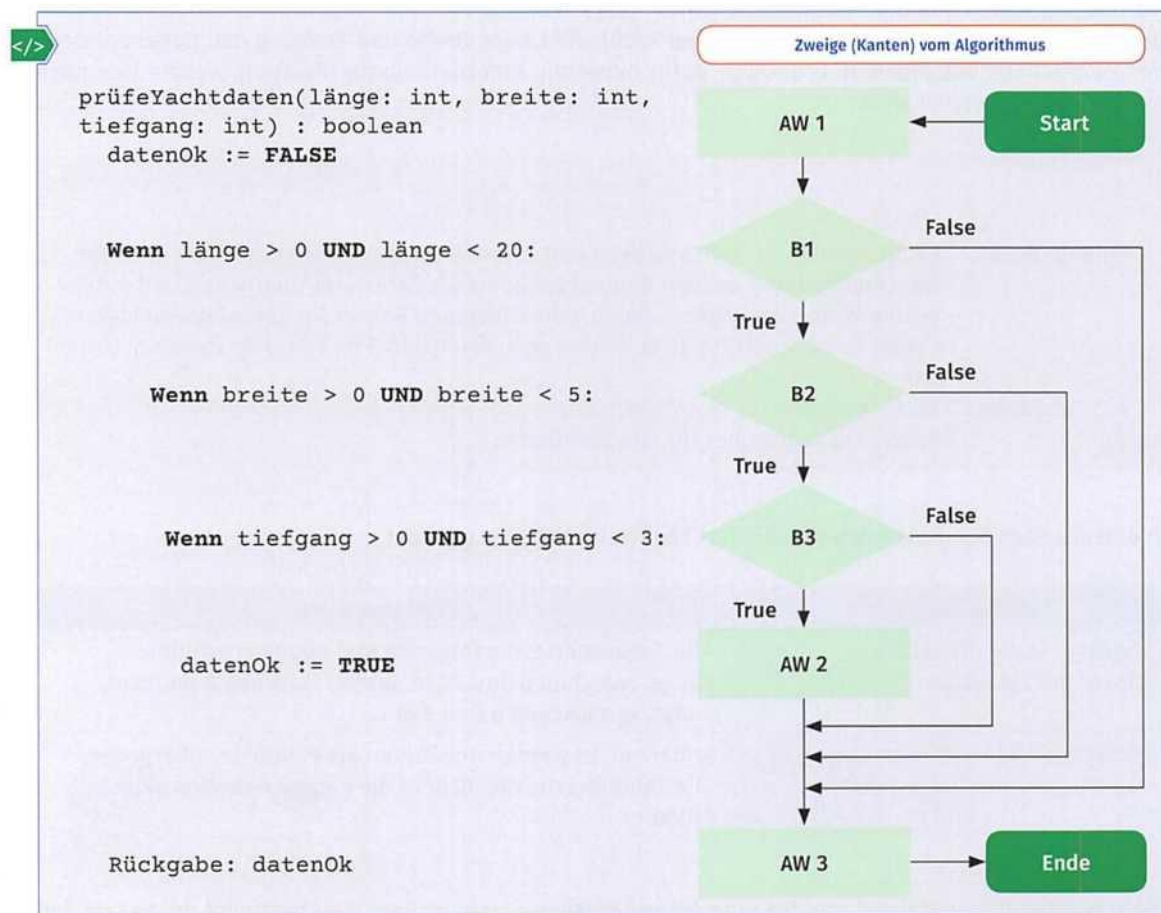
Diese werden nun verfeinert und für jede Äquivalenzklasse werden Testdaten bestimmt. In diesem Fall bieten sich die Grenzwerte an, da hier die meisten Fehler auftreten.

Äquivalenzklasse	Wertebereich	Testdaten (Grenzwerte)
1.1 gültige Äquivalenzklasse	Länge: zwischen 0 und 20	Länge: 0.01 und 19.99
1.2 gültige Äquivalenzklasse	Breite: zwischen 0 und 5	Breite: 0.01 und 4.99
1.3 gültige Äquivalenzklasse	Tiefgang: zwischen 0 und 3	Breite: 0.01 und 2.99
2.1 ungültige Äquivalenzklasse	Länge: kleiner gleich 0	Länge: 0
2.2 ungültige Äquivalenzklasse	Länge: größer gleich 20	Länge: 20
2.3 ungültige Äquivalenzklasse	Breite: kleiner gleich 0	Breite: 0
2.4 ungültige Äquivalenzklasse	Breite: größer gleich 5	Breite: 5
2.5 ungültige Äquivalenzklasse	Tiefgang: kleiner gleich 0	Tiefgang: 0
2.6 ungültige Äquivalenzklasse	Tiefgang: größer gleich 3	Tiefgang: 3

(2) Beispiel für die Ermittlung der Testdaten für einen White-Box-Test

Um für die gleiche Aufgabe Testdaten für einen White-Box-Test zu ermitteln, muss der Algorithmus bekannt sein.

Hier in diesem Beispiel wird ein programmiersprachenunabhängiger Pseudocode verwendet. Eine mögliche Variante für den Algorithmus ist nachfolgend dargestellt.



Für eine Anweisungsüberdeckung reicht es, wenn jede Anweisung einmal durchlaufen wird. Für diesen Algorithmus reichen in diesem Fall Übergebeparameter, welche innerhalb der geforderten Länge, Breite und Tiefgang liegen.

Testdaten für eine Anweisungsüberdeckung: z.B. Länge: 10.00, Breite: 2.50, Tiefgang: 1.20

Bei der Zweigüberdeckung ist das Ziel, dass jeder Zweig (Kante) einmal durchlaufen wird. Dieses betrifft auch nicht vorhandene „Else“-Zweige. Alle möglichen Zweige vom Algorithmus sind im nachfolgenden Programmablaufplan dargestellt. Hier werden bereits mehr Testdaten benötigt.

Zweig	Testdaten
AW1->B1->B2->B3->AW2->AW3	z.B. Länge: 10.00, Breite: 2.50, Tiefgang: 1.20 (siehe auch Anweisungsüberdeckung)
AW1->B1->AW3	z.B. Länge: 40.00, Breite: 2.50, Tiefgang: 1.20
AW1->B1->B2->AW3	z.B. Länge: 10.00, Breite 7.00, Tiefgang: 1.20
AW1->B1->B2->B3->AW3	z.B. Länge: 10.00, Breite: 2.50, Tiefgang: 3.60

✓ Kompetenzcheck

- Geben Sie an, welche Aussagen richtig sind.
 - Die Grenzwertanalyse ist ein typisches Verfahren, um Testfälle für einen White-Box-Test zu ermitteln.
 - Bei einem Black-Box-Test werden die Testfälle u.a. in sogenannte Äquivalenzklassen eingeteilt und hieraus relevante Testfälle ausgewählt.
 - Die komplette Zweigüberdeckung schließt immer eine komplette Anweisungsüberdeckung mit ein.
 - Ein Vorteil der Zweigüberdeckung ist, dass nicht erreichbare Programmteile gefunden werden.
 - Eine Anweisungsüberdeckung ist schwer zu realisieren.
 - Es werden durch eine vollständige Anweisungsüberdeckung mehr Fehler gefunden als durch eine vollständige Zweigüberdeckung.

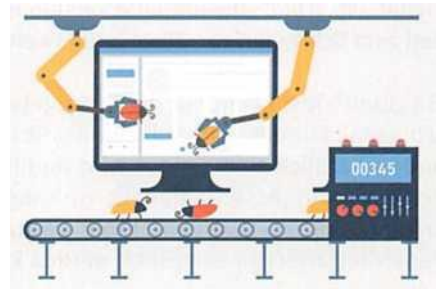
2. Bearbeiten Sie im Arbeitsbuch die Aufgabe 2 der Lernsituation 4 und planen Sie für verschiedene Szenarien die Testfälle.

4.3. Automatisierte Unittests entwerfen, implementieren und anwenden

➤ Sie sollen automatisierte Unittests in einer Programmiersprache umsetzen.

Modultests gehören zu den grundlegendsten Teststufen. Diese Tests werden angewendet, um einzelne Funktionen einer Software zu testen. Im Rahmen des objektorientierten Programmierparadigmas trifft dieses beispielsweise auf die Überprüfung der Funktionalität von Klassen zu.

In agilen Vorgehensmodellen wie Serum wird zur Qualitätssicherung eine sehr häufige Testausführung angestrebt. Diese Vorgehensweise lässt sich nur erreichen, wenn die Tests vollständig automatisiert werden. Um Modultests zu automatisieren, werden diese als eigenständige Programmteile implementiert. So können sie schnell und unkompliziert jederzeit ausgeführt werden. Dieses geschieht in Form von automatisierten Unittests.



Ist ein Unittest implementiert, kann er angewendet werden. Dazu führen Entwickler, wenn sie den Quellcode der Komponente verändern möchten, zunächst den Unittest aus und stellen damit sicher, dass der Code korrekt ausgeführt wird und ein richtiges Testergebnis liefert. Nach der Änderung des Quellcodes wird wiederum der Unittest ausgeführt. Misslingt dieser, dann enthält der angepasste Quellcode Fehler und muss korrigiert werden (siehe auch Regressionstests). Um gute und gebrauchsfähige Unittests zu entwickeln, sollten diese einigen grundlegenden Kriterien genügen. Die folgende Auflistung gibt in dieser Hinsicht einige Anregungen.

Merkmale eines guten Unittests

Die Basis für einen guten Unittest ist das Erfüllen der sogenannten F.I.R.S.T-Prinzipien.

1. **Fast:** Die Testausführung soll möglichst schnell sein, damit sie möglichst oft ausgeführt werden kann. Ideal eignen sich dazu vollständig automatisierte Tests.
2. **Independent:** Unittests sollen unabhängig voneinander sein, damit sie in beliebiger Reihenfolge, parallel oder einzeln ausgeführt werden können.
3. **Repeatable:** Führt man einen Unittest mehrfach aus, muss er immer das gleiche Ergebnis liefern.
4. **Self-Validating:** Ein Unittest soll entweder fehlschlagen oder gut gehen. Diese Entscheidung muss der Test treffen. Es dürfen keine weiteren manuellen Prüfungen nötig sein.
5. **Timely:** Man soll Unittests vor der Entwicklung des Produktivcodes schreiben.

Weitere Kriterien sind:

6. **Begrenztheit:** Es wird pro Test immer nur eine Eigenschaft oder Einheit getestet, z. B. eine Methode.
7. **Verständlichkeit:** Der Quelltext der Tests und dessen Inhalt sind leicht verständlich und so kurz und knapp wie möglich gehalten.
8. **Wartbarkeit:** Der Quelltext der Tests hat auch eine hohe Codequalität, z. B. wurden die üblichen Programmierkonventionen eingehalten.
9. **Relevanz:** Es werden nur notwendige Programmteile getestet.

Fast alle modernen Programmiersprachen bieten Möglichkeiten zum Schreiben automatischer Unittests an. In der Regel ist es üblich, aber nicht zwingend notwendig, dass der Unittest in der gleichen Sprache wie das Testobjekt geschrieben wird. Nachfolgend wird die Entwicklung von Unittests exemplarisch für Java und Python aufgezeigt.

(1) Unittests unter Java

Ein Framework zum Testen von Java-Programmen, welches sich besonders für automatisierte Unittests eignet, ist „JUnit“. Die aktuelle Version ist „JUnit 5“. Das mit „JUnit 5“ vorgestellte neue Programmiermodell zum Schreiben von Tests und Erweiterungen trägt den Namen „JUnit Jupiter API“.

Ein „JUnit“-Test kennt nur die Testergebnisse grün (der Test war erfolgreich) oder rot (der Test ist fehlgeschlagen). Für das Fehlschlagen des Tests können wiederum Fehler (error) oder falsche Ergebnisse (failure) ursächlich sein. Beides wird mithilfe von Exceptions angezeigt. Technisch werden Failures mittels der Exception „AssertionFailedError“ angezeigt, während alle übrigen Exceptions vom JUnitframework als Error interpretiert werden. „JUnit“ besitzt verschiedene Annotations und Assertions, welche zu verschiedenen Testzwecken eingesetzt werden können. Einige wichtige sind nachfolgend aufgeführt.

Annotation	Beschreibung
@Test	legt eine Methode als Testmethode fest
@Timeout(<time>) @Test	Testmethode, die nach time ms fehlschlägt
@BeforeEach	vor jedem Test ausführen
@AfterEach	nach jedem Test ausführen
@BeforeAU	einmal zu Beginn ausführen
@AfterAU	einmal zum Ende ausführen
@DisplayName	in der Testübersicht wird der Text von DisplayName angezeigt anstatt der Methodenname

Assertion	Beschreibung
assertTrue(pred)	prüft, ob Prädikat zu True ausgewertet wird
assertFalse(pred)	prüft, ob Prädikat zu False ausgewertet wird
assertEquals(expected, actual)	prüft, ob Actual und Expected gleich sind; ist dieses nicht der Fall, schlägt der Test fehl
fail(message)	lässt den Test mit der Fehlermeldung „message“ fehlschlagen

Am besten lässt sich die Umsetzung eines Unittests in Java anhand eines Beispiels erläutern.

Beispielaufgabe

In dem Beispiel bildet die Methode „calculateSum“ die Summe aus zwei ganzen Zahlen. Die Testdaten werden ermittelt, z. B. mithilfe der Anweisungsüberdeckung oder der Äquivalenzklassenbildung, und für den Test genutzt. In diesem Fall sind es die Testdaten 3 und 1 und es wird ein Ergebnis von 4 erwartet. Das Ergebnis wird mit calculateSum ermittelt und danach mithilfe der Assertion „assert-Equals“ mit dem erwarteten Ergebnis verglichen. Falls das Ergebnis nicht mit dem erwarteten Ergebnis übereinstimmt, dann erfolgt eine entsprechende Ausgabe. Normalerweise wird der Code für den Unittest von dem zu testenden Code getrennt. Für das folgende Beispiel wurde aber darauf verzichtet und alles in eine Klasse geschrieben.

```

</> import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

class BeispielJUnitTest {

    @Test
    void testCalculateSum () {
        int testValue1 = 3;
        int testValue2 = 1;
        int expectedResult = 4;
        int result = calculateSum(testValue1, testValue2);
        assertEquals("Summe falsch", expectedResult, result);
    }

    public static int calculateSum(int x, int y) {
        return x + y;
    }
}

```

Diagramm zur Verdeutlichung:

- Ein Pfeil weist von der Beschriftung **Unittest** auf die Methode `testCalculateSum` innerhalb der Klasse `BeispielJUnitTest`.
- Ein Pfeil weist von der Beschriftung **zu testende Methode** auf die statische Methode `calculateSum` innerhalb der Klasse `BeispielJUnitTest`.

(2) Unittests unter Python

Zum Entwickeln von automatischen Unittests wird in Python das Modul „unittest“ verwendet, welches sich in der Standardbibliothek von Python befindet. Dieses Modul lehnt sich im Aufbau und der Handhabung u.a. an „JUnit“ an. Es definiert eine Klasse „TestCase“, mit der die Tests implementiert werden. Ein Unittest in Python kann folgende drei Ergebnisse haben:

Ergebnis	Beschreibung
OK	wenn alle Tests erfolgreich verlaufen sind
Failure	wenn einer von den Tests fehlschlägt und ein AssertionError auftritt
Error	wenn ein anderer Fehler als ein AssertionError auftritt

In Python können verschiedene Assertions zu verschiedenen Testzwecken eingesetzt werden. Einige wichtige sind nachfolgend aufgeführt:

Assertion	Beschreibung
<code>assertEqual(arg1, arg2, msg = None)</code>	testet, ob <i>arg1</i> und <i>arg2</i> gleich sind; wenn sie nicht gleich sind, schlägt der Test fehl
<code>assertNotEqual(arg1, arg2, msg = None)</code>	testet, ob <i>arg1</i> und <i>arg2</i> nicht gleich sind; wenn sie gleich sind, schlägt der Test fehl
<code>assertTrue(expr, msg = None)</code>	testet, ob <i>expr</i> wahr ist; wenn nicht, dann schlägt der Test fehl
<code>assertFalse(expr, msg = None)</code>	testet, ob <i>expr</i> falsch ist; wenn nicht, dann schlägt der Test fehl
<code>assertGreater (first, second, msg = None)</code>	testet, ob <i>first</i> größer als <i>second</i> ist; wenn nicht, dann schlägt der Test fehl
<code>assertLess (first, second, msg = None)</code>	testet, ob <i>first</i> kleiner als <i>second</i> ist; wenn nicht, dann schlägt der Test fehl

Am besten lässt sich die Umsetzung eines Unittests in Python anhand eines Beispiels erläutern.

Beispielaufgabe

Es soll eine Funktion, welche das Volumen eines Würfels berechnet, getestet werden. Die Testdaten werden z.B. mithilfe einer Anweisungsüberdeckung oder einer Grenzwertanalyse gebildet. Für das Beispiel wurden vier Testfälle ausgewählt, wobei beispielsweise der Testfall mit 0 einen Grenzwert darstellt. Die Ergebnisse für die einzelnen Testdaten werden durch den Aufruf der Funktion ermittelt und danach mithilfe von „assertAlmostEqual“ mit dem erwarteten Wert verglichen. Für das Beispiel wurden sowohl der Test als auch die Funktion in eine Datei geschrieben. In der Praxis werden die Tests in der Regel von dem zu testenden Code getrennt. Um den Test auszuführen, ist die Datei unter „PyCharm“ als Unittest auszuführen.

```

import unittest

class Example_Unittest(unittest.TestCase):
    def test_calculate_cube_volume(self):
        self.assertEqual(calculate_cube_volume(2), 8)
        self.assertEqual(calculate_cube_volume(1), 1)
        self.assertEqual(calculate_cube_volume(0), 0)
        self.assertEqual(calculate_cube_volume(1.5), 3.375)

def calculate_cube_volume(x):
    return (x * x * x)

```

Unitest

zu testende Methode

Ausgabe

Das Ergebnis eines erfolgreichen Tests sieht dann wie folgt aus:

```

Launching unittests with arguments python -m unittest C:/Python_Projekte/Testum/test.py in C:/Python_Projekte/Testum

Ran 1 test in 0.001s

OK

```

✓ Kompetenzcheck

1. Geben Sie an, welche Aussagen richtig sind.
 - a. Dynamische Testverfahren lassen sich gut automatisieren.
 - b. Auch bei der Entwicklung von Unittests sollte auf wartbaren Code geachtet werden.

- c. Gute Unittests zeichnen sich durch eine hohe Abhängigkeit der einzelnen Tests untereinander aus.
 - d. Ein guter Unittest sollte immer schnell durchzuführen sein.
 - e. Automatische Unittests müssen immer in der gleichen Sprache wie die zu testende Komponente geschrieben sein.
 - f. Das Framework zum Testen für Java-Programme nennt sich „Java-Unit“.
 - g. Das Modul zum Schreiben von Unittests in Python heißt „unittest“, gehört zur Standardbibliothek von Python und wird mit „import“ eingebunden.
2. Bearbeiten Sie im Arbeitsbuch die Lernsituation 4 mit der Aufgabe 3.
 3. Überprüfen Sie in Prüfungsvorbereitung Ihr Wissen über das Testen von Software mithilfe der Aufgabe 4 der Lernsituation 4.

5. Softwarekomponenten und deren Schnittstellen dokumentieren

► Sie sollen Softwarekomponenten und deren Schnittstellen dokumentieren.

Software sollte aus Komponenten bestehen, die über Schnittstellen miteinander kommunizieren. Diese Komponenten können sowohl interne als auch externe Schnittstellen besitzen. Auf der einen Seite ist die Dokumentation der Komponente selbst für eventuelle spätere Wartung und Erweiterung von Bedeutung, auf der anderen Seite ist eine ausführliche Dokumentation der Schnittstellen dieser Komponenten sehr wichtig.



(1) Dokumentation von Softwarekomponenten

Die Dokumentation einer Softwarekomponente zählt zur Entwicklungsdokumentation (siehe auch Jahrgangsband 1, Lernfeld 5). In der Regel sollte die Implementation einer Komponente so wenig wie möglich mit Kommentaren versehen werden und am besten selbsterklärend sein (siehe auch Clean Code). Allerdings gehen auch hier die Meinungen auseinander. Es gibt Entwickler, die davon überzeugt sind, dass guter Code überhaupt keiner Kommentare und Beschreibungen bedarf, während andere eine ausführliche Kommentierung des Quellcodes durchführen. Zumindest sollte im Entwicklungsteam in dieser Beziehung ein einheitliches Vorgehen angestrebt werden. Besser als der Quellcode beschreiben Struktogramme, Klassen-, Anwendungsfall- oder Aktivitätsdiagramme die Komponenten und ihre Funktionalität (siehe auch Kapitel 2.1.2). Allerdings müssen diese bei Änderungen und Erweiterungen der Komponente immer aktuell gehalten werden. Auch die Schnittstellen der Komponente werden in diese Dokumentation aufgenommen, allerdings aus Entwicklersicht und nicht aus der Sicht des Benutzers der Komponente.

(2) Dokumentation der Schnittstellen von Softwarekomponenten

Ein weiterer wichtiger Punkt bei der Dokumentation von Softwarekomponenten ist die Dokumentation der Schnittstellen. Benutzer einer Softwarekomponente, egal ob diese intern verwendet oder öffentlich zur Verfügung gestellt wird, benötigen eine vollständige, aktuelle Dokumentation von deren Schnittstellen. Das Ziel einer guten Schnittstellendokumentation besteht darin, Entwicklern und Testern ausreichend Informationen für ihre Arbeit zu geben.

Inhalte einer Schnittstellendokumentation

Übersicht über alle Schnittstellen einer Softwarekomponente

Zunächst müssen alle Schnittstellen einer Softwarekomponente übersichtlich dargestellt werden. Am besten werden hier grafische Darstellungen verwendet, z.B. in Form von UML-Komponentendiagrammen. Aber auch eine systematische Gliederung von Softwarekomponenten und deren Schnittstellen ist durchaus gebräuchlich.

Funktionale Eigenschaften

In diesem Schritt werden die einzelnen, nach außen verfügbaren Funktionen einer Schnittstelle genauer beschrieben. Dabei sollten für jede Funktion folgende Informationen zur Verfügung gestellt werden:

- Name und Zweck der Funktion
- Name, Bedeutung und Wertebereiche von Übergabe- und Rückgabeparametern
- Verhalten der Komponente bei Nutzung dieser Funktion
- wichtige Zusatzinformationen wie verwendete Formeln oder Normen

Nicht funktionale Eigenschaften

Hier werden sowohl für die gesamte Softwarekomponente als auch für einzelne Funktionen der Schnittstelle folgende nicht funktionale Eigenschaften beschrieben:

- Effizienz

Beispiele: Wie schnell reagiert die Softwarekomponente auf Aufrufe über die Schnittstelle? Wie ändert sich die Bearbeitungsgeschwindigkeit in Abhängigkeit von der Anzahl der Aufrufe pro Zeiteinheit? Welche Datenmenge kann pro Zeiteinheit verarbeitet werden?

- Informationssicherheit

Beispiele: Werden die Daten verschlüsselt übertragen? Muss man sich zur Benutzung der Schnittstelle authentifizieren und wenn Ja wie?

- Zuverlässigkeit

Beispiele: Wie reagiert die Funktion auf fehlerhafte Eingaben? Was sind fehlerhafte Eingaben? Wie reagiert die Softwarekomponente auf eine unterbrochene Datenübertragung?

Heutzutage werden solche Dokumentationen nur noch selten in Printform zur Verfügung gestellt. Vielmehr dominiert hier die digitale Form der Dokumentation, welche einige Vorteile bietet (siehe auch Kapitel 1.6).

Auch das Erstellen von Schnittstellendokumentationen wird immer mehr automatisiert. Dazu kommen Dokumentationswerkzeuge zum Einsatz, welche aus verschiedenen Projektunterlagen Dokumentationen erzeugen. Im Zusammenhang mit Schnittstellendokumentationen wird meistens aus „speziellen“ Kommentaren innerhalb des Quellcodes die Dokumentation generiert. Nachfolgend werden einige dieser Werkzeuge vorgestellt.

Dokumentationswerkzeuge

Javadoc

<https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html>

Javadoc ist ein Dokumentationswerkzeug, welches aus Java-Quelltexten automatisch HTML-Dokumentationsdateien erstellt, und ist Bestandteil des Java Development Kits. Zur automatischen Generierung der Dokumentation müssen in dem Quelltext spezielle Kommentare geschrieben werden.

```
</> /**
 * Beschreibung der Funktion
 * @param paramter1
 * @return
 * @throws IOException
 */
```

Durch verschiedene Tags, z.B. @return, können Klassen, Interfaces und Funktionen näher beschreiben werden. Hieraus werden dann mithilfe von Javadoc entsprechende HTML- Dokumente generiert.

Die Dokumentation der Java-API wurde mithilfe von Javadoc erstellt, [https:// docs.oracle.com/en/java/javase/17/docs/api/](https://docs.oracle.com/en/java/javase/17/docs/api/).

In der Abbildung ist ein Teil der Dokumentation der Scanner-Klasse zu sehen.



Sphinx

<https://www.sphinx-doc.org/en/master/>

Sphinx ist ein Dokumentationswerkzeug, welches in Python geschrieben ist und automatisch aus Python-Quelltexten HTML-Dokumentationsdateien erstellt. Sphinx wurde zunächst zur Dokumentation von Python-Programmen geschrieben. Heutzutage wird Sphinx auch in anderen Umgebungen verwendet. Unter dem oben angegebenen Link ist eine ausführliche Einführung in das Dokumentationswerkzeug zu finden.

Zur Dokumentation im Quellcode wird reStructuredText (ReST, reST oder RST) genutzt. Hierbei handelt es sich um eine vereinfachte Auszeichnungssprache. Beispiele für Syntaxelemente von RST sind in folgender Abbildung zu sehen.

8.8. Class definitions

A class definition defines a class object (see section [The standard type hierarchy](#)):

```
classdef ::= [decorators] "class" classname [inheritance] ":" suite
inheritance ::= "(" [argument_list] ")"
classname ::= identifier
```

A class definition is an executable statement. The inheritance list usually gives a list of base classes (see [Metaclasses](#) for more advanced uses), so each item in the list should evaluate to a class object which allows subclassing. Classes without an inheritance list inherit, by default, from the base class `object`; hence,

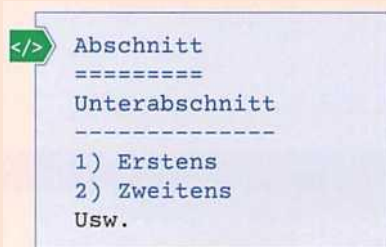
```
class Foo:
    pass
```

is equivalent to

```
class Foo(object):
    pass
```

Die Dokumentation der Python-API wurde auch mithilfe von Sphinx erstellt <https://docs.python.org/3.11/reference/>.

In der Abbildung ist ein Teil dieser Dokumentation zu sehen.



```
</> Abschnitt
=====
Unterabschnitt
-----
1) Erstens
2) Zweitens
Usw.
```

Doxygen

<https://www.doxygen.nl/index.html>

Doxygen ist ein freies Dokumentationswerkzeug unter der GNU General Public License. Auch hier werden im Quellcode spezielle Kommentare geschrieben, aus denen dann eine HTML-Dokumentation generiert wird. Doxygen bildet u.a. die Vererbungshierarchie und die Zusammenhänge zwischen verschiedenen Klassen bei Objektorientierten Programmen sehr gut ab und kann in vielen Programmiersprachen eingesetzt werden. Zu nennen sind hier vor allem C++, C oder Objective-C. Aber auch die Verwendung in Java oder Python ist möglich.

✓ Kompetenzcheck

1. Geben Sie an, welche Aussagen richtig sind.
 - a. Schnittstellen von Softwarekomponenten müssen nicht dokumentiert werden.
 - b. Der interne Code von Software Komponenten sollte so weit wie möglich selbsterklärend sein.
 - c. Export- und Importfunktionen zählen zu den funktionalen Eigenschaften einer Schnittstelle.
 - d. Von den Funktionen einer Softwareschnittstelle sollten u.a. die Übergabeparameter und die Rückgabewerte dokumentiert werden.
 - e. Nicht funktionale Eigenschaften spielen bei der Dokumentation von Schnittstellen keine Rolle.
 - f. Doxygen kann zur automatischen Dokumentationsgeneration von Java-Programmen eingesetzt werden.
 - g. Javadoc ist Bestandteil vom Java Development Kit.
 - h. Sphinx wurde ursprünglich zur automatischen Dokumentationsgeneration von Python-Programmen entwickelt.
2. Bearbeiten Sie im Arbeitsbuch von der Lernsituation 4 die Aufgaben 5 und 6. Überprüfen Sie damit u.a. Ihr Wissen über die Dokumentation von Software.