

# FI - Lernfeld 5

---

## Datenstrukturen für die Softwareentwicklung auswählen und anwenden

Lernfeld	Bildungsgang	Ausbildungsjahr
5: Software zur Verwaltung von Daten anpassen	Fachinformatiker/-in (FI)	1
Kompetenzformulierung		
Curricularer Bezug	Titel der Lernsituation (Kurzfassung)	Geplanter Zeitrichtwert

Lernfeld 5: Software zur Verwaltung von Daten anpassen      LS 5.1: Datenstrukturen für die Softwareentwicklung auswählen und anwenden      6 Unterrichtsstunden

### Handlungssituation

Ein mittelständisches Softwareunternehmen entwickelt eine neue Anwendung zur Verwaltung von Kundendaten und Bestellvorgängen. Als angehende Fachinformatikerin bzw. angehender Fachinformatiker wurden Sie in das Entwicklungsteam aufgenommen und mit der Auswahl geeigneter Datenstrukturen für verschiedene Komponenten der Anwendung betraut.

Die Projektleitung hat folgende Anforderungen formuliert:

- Verwaltung einer dynamisch wachsenden Liste von Kundenkonten
- Schnelle Suche nach Kunden anhand ihrer Kundennummer
- Verwaltung einer Warteschlange für eingehende Bestellungen (First-In-First-Out)
- Speicherung des Bearbeitungsverlaufs mit Rückgängig-Funktion (Last-In-First-Out)
- Sortierte Speicherung von Produktkategorien ohne Duplikate
- Effiziente Zuordnung von Produktcodes zu Produktinformationen

Sie sollen analysieren, welche grundlegenden Datenstrukturen (Arrays, Listen, Stacks, Queues, Sets, Maps) für die jeweiligen Anforderungen am besten geeignet sind. Dabei berücksichtigen Sie Performance-Aspekte (Zeitkomplexität), Speicherverbrauch und Wartbarkeit. Ihre Empfehlungen sollen dem Team helfen, die richtige Grundlage für eine effiziente und skalierbare Software zu schaffen.

### Handlungsergebnis

Die Schülerinnen und Schüler erstellen eine technische Dokumentation mit:

- Übersicht über grundlegende Datenstrukturen (Array, Liste, Stack, Queue, Set, Map/Dictionary)
- Vergleichstabelle mit Eigenschaften, Operationen und Zeitkomplexität je Datenstruktur
- Zuordnung geeigneter Datenstrukturen zu den konkreten Anforderungen mit Begründung
- Beispielhafte Code-Implementierungen in einer Programmiersprache (z.B. Java, Python, C#)
- Darstellung typischer Anwendungsfälle und Best Practices

## Vorausgesetzte Fähigkeiten und Kenntnisse

	<b>Handlungskompetenz (Fachkompetenz und Personale Kompetenz)</b>	<b>Inhalte</b>	<b>Sozialform/Methoden</b>
Informieren bzw. Analysieren	Die Schülerinnen und Schüler recherchieren systematisch Informationen zu grundlegenden Datenstrukturen und analysieren deren Eigenschaften. Sie unterscheiden zwischen linearen und nichtlinearen Datenstrukturen sowie zwischen statischen und dynamischen Speicherstrukturen.	<ul style="list-style-type: none"> <li>- Grundlegende Datenstrukturen (Array, Liste, Stack, Queue, Set, Map)</li> <li>- Eigenschaften und Operationen</li> <li>- Zeitkomplexität (Big-O-Notation)</li> <li>- Grundlagen</li> <li>- Speicherverwaltung</li> <li>- Vor- und Nachteile verschiedener Strukturen</li> <li>- Anwendungsfälle</li> </ul>	<ul style="list-style-type: none"> <li>Einzelarbeit,</li> <li>Partnerarbeit</li> <li>Internetrecherche</li> <li>Fachliteratur</li> <li>Video-Tutorials</li> <li>Quellenanalyse</li> </ul>
Planen / Entscheiden	Die Schülerinnen und Schüler wägen Vor- und Nachteile verschiedener Datenstrukturen für konkrete Anforderungen ab und treffen begründete Auswahlentscheidungen basierend auf Performance, Speicherbedarf und Einsatzzweck.	<ul style="list-style-type: none"> <li>- Anforderungsanalyse</li> <li>- Auswahlkriterien</li> <li>(Performance, Speicher, Komplexität)</li> <li>- Zuordnung</li> <li>Datenstruktur zu Anwendungsfall</li> <li>- Abwägung von Trade-offs</li> <li>- Begründung der Entscheidung</li> </ul>	<ul style="list-style-type: none"> <li>Gruppenarbeit</li> <li>Entscheidungsmatrix</li> <li>Diskussion im Team</li> <li>Peer-Feedback</li> </ul>
Durchführen	Die Schülerinnen und Schüler implementieren ausgewählte Datenstrukturen in einer Programmiersprache und erstellen Beispielcode für typische Operationen. Sie dokumentieren ihre Implementierungen nachvollziehbar.	<ul style="list-style-type: none"> <li>- Implementierung von Datenstrukturen</li> <li>- Verwendung von Standard-Bibliotheken</li> <li>- Code-Beispiele für CRUD-Operationen</li> <li>- Kommentierung und Dokumentation</li> <li>- Testfälle für Grundoperationen</li> <li>- Debugging</li> </ul>	<ul style="list-style-type: none"> <li>Einzel-/Partnerarbeit</li> <li>Pair Programming</li> <li>Code-Review</li> <li>Entwicklungsumgebung (IDE)</li> <li>Versionsverwaltung</li> </ul>

	<b>Handlungskompetenz (Fachkompetenz und Personale Kompetenz)</b>	<b>Inhalte</b>	<b>Sozialform/Methoden</b>
Kontrollieren / Bewerten	Die Schülerinnen und Schüler überprüfen ihre Implementierungen auf Korrektheit und bewerten die Performance ihrer gewählten Datenstrukturen. Sie testen Edge Cases und validieren die Funktionalität.	- Funktionstest der Implementierung - Performance-Messungen - Code-Qualitätsprüfung - Überprüfung der Anforderungserfüllung - Fehleranalyse	Partnerarbeit Unit-Testing Code-Review Testprotokolle
Reflektieren	Die Schülerinnen und Schüler reflektieren ihre Auswahlentscheidungen und erkennen die Bedeutung der richtigen Datenstrukturwahl für die Softwarequalität. Sie übertragen ihr Wissen auf andere Entwicklungsszenarien und bewerten alternative Lösungsansätze.	- Reflexion der Entscheidungsprozesse - Bedeutung für Softwarequalität - Alternative Lösungsansätze - Lessons Learned - Transfer auf andere Projekte - Selbsteinschätzung der erworbenen Kompetenz	Plenum Reflexionsgespräch Retrospektive Selbstreflexion Feedback-Runde

## Arbeitsmaterialien / Links

- Fachliteratur zu Algorithmen und Datenstrukturen
- Online-Dokumentationen der verwendeten Programmiersprachen (Java API, Python Docs, C# Docs)
- Visualisierungstools für Datenstrukturen (z.B. VisuAlgo, Data Structure Visualizations)
- Entwicklungsumgebungen (IntelliJ IDEA, Visual Studio Code, PyCharm, Eclipse)
- Online-Tutorials und Lernplattformen (z.B. W3Schools, GeeksforGeeks, Codecademy)
- GitHub für Code-Sharing und Versionsverwaltung
- Big-O Cheat Sheet zur Komplexitätsanalyse
- Stack Overflow für Problemlösungen und Best Practices
- Whiteboard-Tools für Diagramme und Entwürfe (z.B. Miro, draw.io)

## Schulische Entscheidungen

### Niveau A (Basis):

- Fokus auf die wichtigsten Datenstrukturen (Array, Liste, Stack, Queue)
- Vorstrukturierte Vergleichstabelle mit vorgegebenen Kategorien
- Geführte Recherche mit konkreten Arbeitsaufträgen und Leitfragen
- Einfache Code-Beispiele mit bereitgestellten Vorlagen
- Verwendung von fertigen Bibliotheksfunktionen (keine eigene Implementierung)

- Zuordnung zu Anwendungsfällen anhand einer vorgegebenen Checkliste
- Kurze praktische Übungen mit klaren Schritt-für-Schritt-Anleitungen

#### Niveau B (Standard):

- Betrachtung aller relevanten Grundstrukturen inkl. Set und Map
- Eigenständige Entwicklung einer Vergleichsübersicht
- Selbstständige Recherche mit groben Themenvorgaben
- Eigene Implementierung einfacher Beispiele mit Standard-Bibliotheken
- Begründete Zuordnung mit Darstellung von Alternativen
- Performance-Überlegungen (qualitativ)
- Strukturierte Dokumentation mit Code-Kommentaren
- Praktische Übungen mit mittlerem Schwierigkeitsgrad

#### Niveau C (Erweitert):

- Vertiefende Analyse inklusive Komplexitätstheorie (Big-O) und Speicherverwaltung
- Entwicklung eines umfassenden Bewertungsmodells
- Vollständig eigenständige Recherche und Quellenauswahl
- Eigene Implementierung von Datenstrukturen (nicht nur Bibliotheksnutzung)
- Differenzierte Empfehlung mit Performance-Analyse und Trade-off-Betrachtung
- Berücksichtigung fortgeschrittenener Strukturen (z.B. Hash-Tabellen, verkettete Listen)
- Professionelle technische Dokumentation mit UML-Diagrammen
- Transfer auf komplexe Anwendungsszenarien
- Vergleich verschiedener Programmiersprachen-Implementierungen (fakultativ)

#### Leistungsnachweise

- Bewertung der Vergleichsübersicht (Vollständigkeit, Korrektheit, Übersichtlichkeit)
- Bewertung der Zuordnung (Begründungstiefe, Schlüssigkeit, Praxisbezug)
- Bewertung der Code-Implementierungen (Funktionalität, Code-Qualität, Dokumentation)
- Praktische Programmieraufgabe (Live-Coding oder Take-Home)
- Mündliche Leistungsüberprüfung durch Fachgespräch oder Code-Review
- Optional: Schriftlicher Test zu Eigenschaften und Zeitkomplexität von Datenstrukturen
- Optional: Präsentation der Ergebnisse mit Demonstration der Implementierung

#### Mögliche Verknüpfungen zu anderen Lernfeldern / Fächern

- **Lernfeld 4:** Einfache IT-Systeme in Betrieb nehmen – Performance und Ressourcenverbrauch verschiedener Datenstrukturen
- **Lernfeld 6:** Serviceanfragen bearbeiten – Warteschlangen für Ticket-Systeme
- **Lernfeld 8:** Datenbanken entwickeln und verwalten – Vergleich mit Datenbankstrukturen und Indizes
- **Lernfeld 9:** Netzwerke und Dienste bereitstellen – Datenstrukturen in Netzwerkprotokollen (Puffer, Queues)
- **Lernfeld 10:** Softwareprojekte planen und durchführen – Datenstruktur-Design in größeren Projekten
- **Mathematik:** Mengentheorie, Graphentheorie (Grundlagen)
- **Englisch:** Technical English, Dokumentation englischsprachiger APIs