Hartmann W., Jäckel S., Näf M., Reichert R.: Informatikunterricht planen und durchführen. 2026, Springer Vieweg Berlin, Heidelberg

Advance Organizer mit generativer KI entwickeln

Wenn es schwierig ist, für ein abstraktes Thema einen passenden Advance Organizer zu finden, kann ein generativer KI-Werkzeug gezielt unterstützen. Dazu gibt man das zentrale Konzept oder den Fachbegriff ein (z. B. Nebenläufigkeit, Polymorphismus, Normalform Datenbank) und bittet um mehrere kurze Erklärungen auf unterschiedlichen Abstraktionsstufen (z. B. alltagsnah, bildhaft, fachlich reduziert. Aus diesen Vorschlägen wählt die Lehrperson eine geeignete Analogie oder Formulierung aus, passt sie an die Lernenden an und nutzt sie als Einstieg, um neues Wissen mit bereits Bekanntem zu verknüpfen.

Die folgenden Beispiele wurden im Januar 2026 mit Google Gemini erstellt und werden in Originalform wiedergegeben.

Beispiel «Nebenläufigkeit»

1. Die Alltags-Ebene (Abstrakt)

Nebenläufigkeit ist wie ein Koch, der gleichzeitig Nudeln kocht und Sauce rührt, indem er ständig zwischen den Aufgaben hin- und herwechselt. Auch wenn er nur zwei Hände hat, werden beide Gerichte fast zeitgleich fertig, weil er Wartezeiten effizient nutzt. In der Informatik bedeutet das, dass ein System mehrere Aufgaben beginnt und bearbeitet, ohne dass eine erst komplett abgeschlossen sein muss. Es geht primär um die Organisation von Aufgaben, nicht zwingend um deren zeitgleiche Ausführung.

2. Die Programmier-Ebene (Anwendung)

Für Entwickler bedeutet Nebenläufigkeit, ein Programm in logisch unabhängige Einheiten wie Threads oder Coroutines zu unterteilen. Diese Einheiten können verschachtelt ausgeführt werden, was die Reaktionsfähigkeit erhöht, etwa damit eine Benutzeroberfläche nicht einfriert, während im Hintergrund Daten geladen werden. Ein Scheduler steuert dabei, welcher Code-Abschnitt wann Rechenzeit erhält. Der Fokus liegt hier auf der sauberen Trennung von Abläufen, die theoretisch auch gleichzeitig ablaufen könnten.

3. Die System-Ebene (Hardware & OS)

Auf der Ebene des Betriebssystems beschreibt Nebenläufigkeit die Verwaltung mehrerer Prozesse, die sich begrenzte Ressourcen wie die CPU teilen. Durch extrem schnelles Hin- und Herwechseln (Context Switching) erzeugt das System die Illusion von Gleichzeitigkeit, selbst wenn nur ein einzelner Prozessorkern vorhanden ist. Dabei müssen Synchronisationsmechanismen wie Locks oder Mutexe verhindern, dass Prozesse gleichzeitig auf denselben Speicher zugreifen und Daten korrumpieren. Hier wird die physikalische Rechenzeit in winzige Zeitscheiben zerlegt.

4. Die Theoretische Ebene (Formal)

In der theoretischen Informatik wird Nebenläufigkeit als die Eigenschaft eines Systems definiert, bei der die zeitliche Reihenfolge von Ereignissen nur partiell geordnet ist. Mathematische Modelle wie Petri-Netze oder Prozesskalküle untersuchen dabei das Verhalten von Systemen,

deren Zustandsübergänge nicht-deterministisch verschachtelt sein können. Es geht darum, zu beweisen, dass ein System trotz beliebiger Abfolgen von Rechenschritten (Interleaving) stets korrekte Ergebnisse liefert und keine Deadlocks erzeugt. Der Fokus liegt auf der Verifikation der logischen Korrektheit komplexer Interaktionen.

Beispiel «Polymorphismus»

1. Die Alltags-Ebene (Abstrakt)

Polymorphismus bedeutet, dass ein und derselbe Befehl je nach Empfänger unterschiedliche Reaktionen auslöst. Denken Sie an das Kommando „Sprechen!“: Ein Mensch antwortet mit Worten, ein Hund bellt und ein Papagei krächzt. In der Informatik ermöglicht dies, dass man verschiedene Dinge auf die gleiche Art ansprechen kann, ohne die genauen Details jedes Einzelnen zu kennen. Es geht um eine einheitliche Schnittstelle für unterschiedliche Verhaltensweisen.

2. Die Programmier-Ebene (Anwendung)

In der objektorientierten Programmierung erlaubt Polymorphismus, dass eine Unterklasse die Methoden ihrer Basisklasse überschreibt. Ein Programm kann zum Beispiel eine Liste von „Formen“ verwalten und für jede einfach zeichne() aufrufen – egal ob es sich um einen Kreis oder ein Quadrat handelt. Der Code muss nicht wissen, welche Form er vor sich hat, da jedes Objekt selbst weiß, wie es sich zeichnen muss. Das macht Software flexibel und leicht erweiterbar für neue Objekttypen.

3. Die System-Ebene (Implementierung)

Technisch wird Polymorphismus oft durch „dynamische Bindung“ (Late Binding) zur Laufzeit realisiert. Der Compiler erstellt dabei häufig eine Sprungtabelle (V-Table), in der die Speicheradressen der tatsächlichen Methoden für den jeweiligen Datentyp hinterlegt sind. Erst wenn das Programm läuft, wird anhand dieser Tabelle entschieden, welcher Codeblock genau ausgeführt werden muss. Dies ermöglicht die Abstraktion von konkreten Datentypen auf der Ebene der Speicherverwaltung.

4. Die Theoretische Ebene (Typ-Theorie)

In der theoretischen Informatik beschreibt Polymorphismus die Fähigkeit eines Typsystems, einen Wert oder eine Funktion unter mehreren Typen zu führen. Man unterscheidet dabei vor allem zwischen Ad-hoc-Polymorphismus (Überladen von Funktionen), Subtyp-Polymorphismus (Vererbung) und parametrischer Polymorphie (Generics). Ziel ist es, abstrakte Algorithmen zu definieren, die über eine Menge von Typen operieren können, während die Typsicherheit mathematisch beweisbar bleibt.

Beispiel «Normalform bei Datenbanken»

1. Die Alltags-Ebene (Abstrakt)

Normalformen sind wie ein striktes Ordnungssystem für einen Aktenschrank, das verhindert, dass dieselbe Information an fünf verschiedenen Stellen abgelegt wird. Jedes Detail bekommt genau einen festen Platz, damit man beim Ändern einer Adresse nicht in zehn verschiedenen Mappen radieren muss. Durch diese Regeln werden doppelte Einträge vermieden, was das Suchen und

Aktualisieren massiv vereinfacht. Das Ziel ist ein „sauberes“ System, in dem keine widersprüchlichen Informationen entstehen können.

2. Die Programmier-Ebene (Anwendung)

Für Entwickler ist die Normalisierung der Prozess, eine große, unübersichtliche Tabelle in mehrere kleinere, logisch verknüpfte Tabellen aufzuteilen. Man trennt beispielsweise Kundendaten von Bestelldaten, um sicherzustellen, dass jede Information nur einmal existiert und über eindeutige Schlüssel (IDs) referenziert wird. Dies verhindert Anomalien, sodass beim Löschen einer Bestellung nicht versehentlich der ganze Kunde aus dem System verschwindet. Die ersten drei Normalformen (1NF, 2NF, 3NF) gelten dabei als Goldstandard für ein robustes Datenbankdesign.

3. Die System-Ebene (Implementierung)

Auf technischer Ebene sichern Normalformen die Datenintegrität und minimieren den Speicherbedarf durch die Vermeidung von Redundanz. Da jede Information nur einmal physisch vorhanden ist, muss das System bei Schreibvorgängen weniger Daten synchronisieren, was die Gefahr von Inkonsistenzen verringert. Allerdings erfordert ein hochgradig normalisiertes System mehr „Joins“ (Zusammenführungen) beim Lesen der Daten, was die Abfragegeschwindigkeit beeinflussen kann. Es geht hier also um die Balance zwischen Datensicherheit und Performance.

4. Die Theoretische Ebene (Formal)

In der relationalen Entwurfstheorie definieren Normalformen Kriterien für die Zerlegung von Relationen basierend auf funktionalen Abhängigkeiten. Ein Schema erreicht eine höhere Normalform, wenn jedes Nicht-Schlüssel-Attribut ausschließlich von einem Primärschlüssel abhängt (getreu dem Motto: „The key, the whole key, and nothing but the key“). Mathematisch wird so sichergestellt, dass die Dekomposition verlustfrei ist und keine transitiven Abhängigkeiten die logische Struktur schwächen. Es ist ein formaler Beweis für die Korrektheit und Redundanzfreiheit eines Datenmodells.