

## Fragen zu Software Engineering allgemein:

### Was ist UML? Nennen Sie die wichtigsten UML-Diagramme!

UML = **Unified Modeling Language**

Die Unified Modeling Language (UML) ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme. Es handelt sich um eine Vereinheitlichung der grafischen Darstellung und Semantik der Modellierungselemente – es wird jedoch keine Methodik beschrieben.

Mit ihrer Hilfe lassen sich Analyse und Design im Softwareentwicklungsprozess beschreiben.

#### Strukturdiagramme

- **Klassendiagramm** (Klassen, Beziehungen untereinander, Merkmale)
- **Komponentendiagramm** (Komponenten, Verdrahtung)
- **Verteilungsdiagramm** (Verteilung von Artefakten auf Knoten)
- Profildiagramm (Stereotypen)
- Objektdiagramm (Exemplare von Klassen mit beispielhaften Werten)
- Kompositionsstrukturdiagramm (Innerer Aufbau von Klassen oder Objekten)
- Paketdiagramm (Ablagestruktur, Abhängigkeiten)
- **Anwendungsfalldiagramm** (Anwendungsfälle, Beziehungen zu Akteuren)

#### Verhaltensdiagramme

- **Aktivitätsdiagramm** (Aktionen, Flüsse, Verzweigungen)
- Interaktionsdiagramm (zulässige Reihenfolge von Aktivitäten)
  - o Interaktionsübersicht (Übersicht über mehrere Interaktionen)
  - o Kommunikationsdiagramm (Topologie des Nachrichtenaustauschs)
  - o **Sequenzdiagramm** (Reihenfolge des Nachrichtenaustauschs)
  - o Zeitdiagramm (zeitlicher Ablauf des Nachrichtenaustauschs)
- Zustandsdiagramm (Zustände, Übergänge, Verhalten)
  - o Protokollautomat (Zulässige Reihenfolge von Operationen)

#### Anwendungsfalldiagramm

Ein Anwendungsfalldiagramm entsteht meist während der Anforderungsphase und beschreibt die Geschäftsprozesse, indem es die Interaktion von Personen – oder von bereits existierenden Programmen – mit dem System darstellt. Die handelnden Personen oder aktiven Systeme werden Akteure genannt und sind im Diagramm als kleine Männchen angedeutet. Anwendungsfälle beschreiben dann eine Interaktion mit dem System.

#### Klassendiagramm

Ein Klassendiagramm stellt zum einen die Elemente der Klasse dar, also die Attribute und Operationen, und zum anderen die Beziehungen der Klassen untereinander. Klassen werden als Rechteck dargestellt, die Beziehungen zwischen den Klassen werden durch Linien angedeutet.

### Sequenzdiagramm

Das Sequenzdiagramm stellt das dynamische Verhalten von Objekten dar. So zeigt es an, in welcher Reihenfolge Operationen aufgerufen und wann neue Objekte erzeugt werden. Die einzelnen Objekte bekommen eine vertikale Lebenslinie, und horizontale Linien zwischen den Lebenslinien der Objekte beschreiben die Operationen oder Objekterzeugungen. Das Diagramm liest sich somit von oben nach unten.

### Einsatz- und Verteilungsdiagramm (Deploymentdiagramm)

Verteilungsdiagramme zeigen, welche Software (Komponenten, Objekte) auf welcher Hardware (Knoten) laufen, d.h. wie diese konfiguriert sind und welche Kommunikationsbeziehungen dort bestehen. Wird vor allem bei Client/Server-Programmen verwendet.

### Komponentendiagramm

Zeigt aus welchen Bausteinen ein Programm besteht. Es zeigt die Organisation und Abhängigkeiten von Komponenten (Definition einer fachlichen Einheit – mit einer internen, aus Klassen oder wiederum Komponenten bestehenden Struktur, die nach außen Funktionalität über Schnittstellen bereitstellt oder anfordert, die unabhängig verwendbar ist).

### Aktivitätsdiagramm

Aktivitätsdiagramm ist eine Beschreibung von komplexen Abläufen. Es beschreibt einen Ablauf und wird definiert durch verschiedene Arten von Knoten, die durch Objekt- und Kontrollflüsse miteinander verbunden sind. Ein Aktivitätsdiagramm besteht aus Start- und Endknoten sowie einer Reihe von Knoten (Aktions-, Kontroll- und Objektknoten) sowie Objekt- und Kontrollflüssen, mit denen diese Knoten verbunden sind.

**Beschreiben Sie die wichtigsten Tätigkeiten, die im Zuge eines Software-Projekts anfallen!**Problemanalyse und Planung

- Ist-Zustand erheben
- Problembereiche abgrenzen
- Geplantes System grob skizzieren

Systemspezifikation

- Pflichtenheft erstellen
- Projektplan genau festlegen
- Systemspezifikation validieren

System- und Komponentenentwurf

## Systemarchitektur entwerfen

- Komponenten definieren
- Schnittstellen entwerfen
- Logisches Datenmodell festlegen
- Algorithmische Struktur entwerfen

Implementierung und Komponententest

- Übertragung in eine Programmiersprache (Codierung)
- Logisches Datenmodell in physisches übertragen
- Übersetzung, statische Prüfungen
- Komponenten testen

Systemtest

- Subsysteme integrieren
- Installation
- Abnahmetest durch Benutzer
- Leistungstests

Betrieb und Wartung

- Im Betrieb entdeckte Fehler beheben
- Systemänderungen und –erweiterungen vornehmen.

ProjektmanagementQualitätsmanagement

**Was verstehen Sie unter Anforderungsanalyse?**

Bei der Anforderungsanalyse arbeiten die Software-Entwickler mit den Kunden und Systemendbenutzern zusammen, um den Anwendungsbereich, die vom System zu leistenden Dienste, die erforderliche Leistungsfähigkeit des Systems, Hardwarebeschränkungen usw. zu bestimmen. Es ist ein sich wiederholender Prozess.

Bei diesem Prozess fallen folgende Aktivitäten an:

- Sammeln der Anforderungen
  - o Grundlagen beschreiben -> Systembeschreibung
  - o Akteure beschreiben -> Aktorliste
  - o Anforderungen finden -> Anwendungsfall-Diagramm
  - o Anforderungen beschreiben -> Anwendungsfall-Beschreibung
  - o Domäne modellieren -> Domänen-Modell/Klassendiagramm
  - o Analyseprototyp entwickeln -> Analyse-Prototyp)
- Klassifizierung und Organisation der Anforderungen (Systemarchitektur)
- Priorisieren der Anforderungen und Auflösung von Konflikten
- Spezifikation der Anforderungen (Dokumentation der Anforderungen)

Systembeschreibung

- Ausgangssituation
- Zielsetzung
- Organisation
- Existierende Systeme in der Branche
- Existierende Systeme im Unternehmen
- Gesetzlicher Rahmen
- Abgrenzung des Systems

**Welche UML-Diagramme können in der Analysephase verwendet werden?**Anwendungsfall-Diagramm (Use Case Diagram)

Dies sind Diagramme mit Ellipsen und Strichfiguren, sie zeigen die strukturellen Zusammenhänge und Abhängigkeiten zwischen Anwendungsfällen und Akteuren. Ein Anwendungsfalldiagramm beschreibt die Zusammenhänge zwischen einer Menge von Anwendungsfällen und den daran beteiligten Akteuren. Sie beschreiben das externe Systemverhalten und die Interaktion der Außenstehenden mit dem System.

Klassendiagramm

Ein Klassendiagramm beschreibt Klassen von Objekten, ihre Eigenschaften, Operationen und Beziehungen zueinander.

Sequenzdiagramm

Das Sequenzdiagramm stellt das dynamische Verhalten von Objekten dar. So zeigt es an, in welcher Reihenfolge Operationen aufgerufen und wann neue Objekte erzeugt werden. Die einzelnen Objekte bekommen eine vertikale Lebenslinie, und horizontale Linien zwischen den Lebenslinien der Objekte beschreiben die Operationen oder Objekterzeugungen. Das Diagramm liest sich somit von oben nach unten.

**Was ist ein Sequenzdiagramm und wofür können Sequenzdiagramm eingesetzt werden?**

Es wird nicht nur der momentane Zustand der Objekte abgebildet, sondern ein ganzer Ablauf. Sie zeigen den Ablauf von Objektinteraktionen.

Das Sequenzdiagramm stellt das dynamische Verhalten von Objekten dar. So zeigt es an, in welcher Reihenfolge Operationen aufgerufen und wann neue Objekte erzeugt werden. Die einzelnen Objekte bekommen eine vertikale Lebenslinie, und horizontale Linien zwischen den Lebenslinien der Objekte beschreiben die Operationen oder Objekterzeugungen. Das Diagramm liest sich somit von oben nach unten.

Im Besonderen steht der zeitliche Verlauf der Kommunikation im Vordergrund. Sequenzdiagramme dürfen als Beteiligte Objekte und weitere Akteure beinhalten. Die Beteiligten kommunizieren über Nachrichten. Sie werden durch ein Rechteck mit Beschriftung, an dem unten eine vertikale, gestrichelte Linie angebracht ist, dargestellt. Die Gesamtheit aus Rechteck und gestrichelter Linie wird als Lebenslinie bezeichnet.

Rechtecke auf den gestrichelten Linien signalisieren, wann eine Lebenslinie aktiv ist. Methodenaufrufe und weitere Kommunikationsmöglichkeiten der Akteure untereinander werden durch Nachrichten dargestellt, die aus einer Beschreibung und einem waagerechten Pfeil bestehen

Am besten verwendet man Sequenzdiagramme zur beispielhaften Veranschaulichung von Vorgängen. Sie bilden nichts Statisches ab, sondern etwas Dynamisches. Sequenzdiagramme werden immer dann gebraucht, wenn man Abläufe innerhalb von Softwaresystemen darstellen will. Mit ihnen ist man in der Lage die Kommunikation zwischen Klassen oder Objekten darzustellen.

**Welche Tätigkeiten fasst man unter dem Begriff „Design“ zusammen?**

Speziell werden alle Tätigkeiten im Rahmen des Softwareentwicklungsprozesses bezeichnet, mit denen ein Modell logisch und physisch strukturiert wird und die dazu dienen zu beschreiben, wie das System die vorhandenen Anforderungen grundsätzlich erfüllt.


- Systemarchitektur entwerfen
- Komponenten definieren
- Schnittstellen spezifizieren
- Definition der Datenstrukturen, Algorithmen
- Kontext und externen Interaktionen des Systems definieren
- Entwurfsmodelle entwickeln

Im Design kümmern sich die Entwickler um die Abbildung der Ideen auf Pakete, Klassen und Schnittstellen.

**Welche Beziehungen zwischen Klassen bzw. Objekten können in einem Klassendiagramm eingetragen werden?**

### Spezialisierung/Generalisierung


Eine Generalisierung/Spezialisierung ist eine Beziehung zwischen einem allgemeineren und einem spezielleren Element (bzw. umgekehrt), wobei das speziellere weitere Eigenschaften hinzufügt und sich kompatibel zum allgemeinen verhält. Vererbung ist ein Umsetzungsmechanismus für die Relation zwischen Ober- und Unterklasse, wodurch Attribute und Operationen der Oberklasse auch den Unterklassen zugänglich gemacht werden.

Notation: Die Vererbungsbeziehung wird mit einem großen, nicht gefüllten Pfeil dargestellt, wobei der Pfeil von der Unterklasse zur Oberklasse zeigt. 

### Abhängigkeit

Eine Abhängigkeit ist eine Beziehung von einem (oder mehreren) Quellelement(en) zu einem (oder mehreren) Zielelement(en).

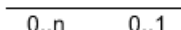
Die Zielelemente sind für die Spezifikation oder Implementierung der Quellelemente erforderlich.

Notation: Dargestellt wird eine Abhängigkeit durch einen gestrichelten Pfeil, wobei der Pfeil vom abhängigen auf das unabhängige Element zeigt. 

### Assoziation

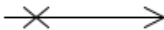
Eine Assoziation beschreibt als Relation zwischen Klassen die gemeinsame Semantik und Struktur einer Menge von Objektverbindungen.

Gewöhnlich ist eine Assoziation eine Beziehung zwischen zwei Klassen. Grundsätzlich kann aber jede beliebige Anzahl von Klassen an einer Assoziation beteiligt sein.

Notation: Assoziationen werden durch eine Linie zwischen den beteiligten Klassen dargestellt. An den jeweiligen Enden der Linie wird die Multiplizität notiert. 

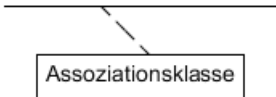
### Gerichtete Assoziation

Eine gerichtete Assoziation ist eine Assoziation, bei der von der einen beteiligten Klasse zur anderen direkt navigiert werden kann, nicht aber umgekehrt.

Notation: Eine gerichtete Assoziation wird wie eine gewöhnliche Assoziation notiert, jedoch hat sie auf der Seite der Klasse, zu der navigiert werden kann eine offene Pfeilspitze. Die Richtung, in die nicht navigiert werden kann, wird durch ein kleines Kreuz auf der Seite der Klasse markiert, zu der nicht navigiert werden kann. 

### Attributierte Assoziation

Eine attributierte Assoziation verfügt sowohl über die Eigenschaften einer Klasse als auch über die einer Assoziation. Es kann gesehen werden als eine Assoziation mit zusätzlichen Klasseneigenschaften oder als Klasse mit zusätzlichen Assoziationseigenschaften. Wenn zwei Klassen in Beziehung zueinander stehen, kann es sein, dass es Eigenschaften gibt, die weder zu einer noch zur anderen Klasse gehören, sondern zur Beziehung zwischen den beiden. Mit einer Assoziationsklasse kann dies modelliert werden.

Notation: Attributierte Assoziationen werden wie gewöhnliche Assoziationen dargestellt, zusätzlich ist jedoch über eine gestrichelte Linie, die von der Assoziationslinie abgeht, ein normales Klassensymbol angehängt. 

### Mehrgliedrige Assoziation

Eine mehrgliedrige Assoziation ist eine Assoziation, an der mehr als zwei Klassen beteiligt sind.

Notation: Eine mehrgliedrige Assoziation wird mit einer nicht ausgefüllten Raute gezeichnet, die größer ist als die Aggregationsraute. Die Klassen werden mit Linien mit der Raute verbunden.



### Qualifizierte Assoziation

Bei einer qualifizierten Assoziation wird die referenzierte Menge der Objekte durch qualifizierende Attribute in Partitionen unterteilt. Die durch eine Assoziation spezifizierte Menge von verlinkten Objekten kann durch eine ihrer Eigenschaften in Untermengen (Partitionen) aufgeteilt werden. Diese Eigenschaft kann man als Qualifizierer modellieren.

Notation: Das für die Assoziation benutzte qualifizierende Attribut wird in einem Rechteck an der Seite der Klasse notiert, die über diesen Qualifizierer auf das Zielobjekt zugreift.



### Aggregation

Eine Aggregation ist eine Assoziation, erweitert um den semantisch unverbindlichen Kommentar, dass die beteiligten Klassen keine gleichwertige Beziehung führen, sondern eine Ganzes-Teile-Hierarchie darstellen. Eine Aggregation soll beschreiben, wie sich etwas Ganzes aus seinen Teilen logisch zusammensetzt. Kennzeichnend für alle Aggregationen ist, dass das Ganze Aufgaben stellvertretend für seine Teile wahrnimmt. Im Gegensatz zur Assoziation führen die beteiligten Klassen keine gleichberechtigten Beziehung, sondern eine Klasse (das Aggregat) bekommt eine besondere Rolle und übernimmt stellvertretend die Verantwortung und Führung.

Notation: Eine Aggregation wird wie eine Assoziation als Linie zwischen zwei Klassen dargestellt und zusätzlich mit einer kleinen Raute versehen. Die Raute steht auf der Seite des Aggregats (des Ganzen).



### Komposition

Eine Komposition ist eine strenge Form der Aggregation, bei der das Ganze verantwortlich ist für die Existenz und Speicherung der Teile. Sie beschreibt, wie sich etwas Ganzes aus Einzelteilen zusammensetzt und diese kapselt.

Notation: Die Komposition wird wie die Aggregation als Linie zwischen zwei Klassen gezeichnet und mit einer kleinen gefüllten Raute auf der Seite des Ganzen versehen.



**Was steht in einem Projektplan? Was ist ein Meilenstein?****Projektplan**

Der Projektplan, der zu Beginn des Projekts erstellt wird, soll das Projektteam und die Kunden über Arbeitsaufteilung und –ablauf informieren und helfen, den Fortschritt des Projekts einzuschätzen. Der Projektplan definiert die Aufteilung der Arbeit und die Ressourcen, die für das Projekt verfügbar sind. Der Plan sollte außerdem die Risiken ermitteln, die dem Projekt und der zu entwickelnden Software drohen, und aufzeigen, welcher Ansatz beim Risikomanagement verfolgt wird.

Der Projektplan umfasst normalerweise folgende Abschnitte:

**Einleitung**

Dies beschreibt kurz die Ziele des Projekts und definiert die Rahmenbedingungen (Finanzrahmen, Zeit), die das Projektmanagement beeinflussen.

**Projektorganisation**

Dieser Abschnitt beschreibt die Organisationsstruktur des Teams, die benötigten Mitarbeiter und ihre Rolle im Team.

**Risikoanalyse**

Diese Analyse beschreibt potenzielle Projektrisiken, ihre Eintrittswahrscheinlichkeit und mögliche Strategien zur Reduzierung der Risiken.

**Anforderungen an Hardware- und Softwareressourcen**

Dieser Abschnitt spezifiziert, welche Hard- und Software zur Entwicklung erforderlich sind. Falls Hardware angeschafft werden muss, können hier Schätzwerte für Preise und Liefertermine angegeben werden.

**Arbeitsaufteilung**

Hier wird das Projekt in Aktivitäten zerlegt und die Meilensteine und Lieferungen der einzelnen Aktivitäten beschrieben.

**Projektzeitplan**

Dieser Abschnitt zeigt die Abhängigkeiten zwischen den Aktivitäten, der geschätzten Zeit zum Erreichen der einzelnen Meilensteine und der Zuweisung von Mitarbeitern zu Aktivitäten.

**Mechanismen zur Überwachung und Berichterstellung**

Dieser Abschnitt definiert welche Berichte das Management zu erstellen hat, wann sie zu erstellen sind und welche Überwachungsmechanismen verwendet werden.

**Meilensteine**

Meilensteine sind wichtige Stadien im Projekt, in denen der Fortschritt bewertet werden kann.

Meilenstein ist ein „Ereignis besonderer Bedeutung“ im Ablauf eines Projekts. Wesentlicher Bestandteil eines Meilensteins ist oft die Termineinhaltung. Man kann einen Meilenstein als einen „nach außen kommunizierten Zeitpunkt, bis wann bestimmte Aufgaben erledigt sein müssen“ verstehen. Ein Meilenstein hat dabei keine zeitliche Ausprägung (Dauer = 0 Tage) und beinhaltet keine Tätigkeit.



**Welche Methoden der Qualitätssicherung können in einem Software-Projekt eingesetzt werden?**Konstruktive Methoden

- Mitarbeiterschulungen
- Angenehme Arbeitsumgebung für Mitarbeiter bereitstellen
- Reviews
- Inspektion (Quellcode inspizieren)
- Tests
- Betaversionen zur Verfügung stellen

Analytische Methoden

- Testverfahren

Testgetriebene Entwicklung

Ein Ansatz der Softwareentwicklung, bei dem ausführbare Tests vor dem Programmcode geschrieben werden. Die Tests laufen automatisch nach jeder Änderung ab, die am Programm vorgenommen werden.

**Was versteht man unter Versionskontrolle?**

Die Änderungen an einem Softwaresystem und seinen Komponenten so zu verwalten, dass es möglich ist nachzuvollziehen, welche Änderungen in einer Version der Komponente bzw. des Systems implementiert wurden, und außerdem vorherige Versionen der Komponente/des Systems wiederherzustellen bzw. erneut zu erzeugen.

**Was beeinflusst den Aufwand eines Software-Projekts? Wie kann Aufwand geschätzt werden?****Einfluss auf den Aufwand**

- Skill-Level der Mitarbeiter
- Erfahrung der Mitarbeiter in der Anwendungsdomäne
- Wie gibt sich der Kunde (ständig neue Anforderungswünsche, ...)
- Strikte gesetzliche Vorgaben ja/nein

**Aufwandschätzverfahren****Empirische Verfahren**

- Analogieverfahren (Erfahrungen aus vergangenen/ähnlichen Projekten)
- Expertenschätzungen (Delphi-Methode)

**Analogieverfahren**

Die Aufwände/Kosten werden auf der Basis der Werte eines oder mehrerer unter denselben Rahmenbedingungen durchgeführter, gleichartiger Projekte geschätzt, d.h. in Analogie zu diesen Projekten.

**Expertenschätzung**

Experten im Bereich der Aufwandschätzung führen Schätzungen durch und diskutieren diese anschließend, um schließlich zu einer konsolidierten Schätzung zu gelangen

**Algorithmische Verfahren**

- Function Point Analyse
- Constructive Cost Model (COCOMO Modell)
- Use Case Point Analyse
- Objekts Point Analyse

**Function Point-Analyse**

Statt der Codegröße wird die Funktionalität des zu entwickelnden Systems geschätzt. Die Anforderungen werden in „Functions“ klassifiziert (Ein-/Ausgabedaten, Referenzdaten, Datenbestände, Abfragen).

**Constructive Cost Model (COCOMO-Mode)**

Auch das COCOMO-Modell setzen Function Points bzw. Object Points ein; Object Points werden Bildschirmmasken, Berichten und Programmmodulen zugeordnet.

## Welche Vorgehensmodelle der Software-Entwicklung gibt es?

Bei einem Vorgehensmodell handelt es sich um eine vereinfachte Darstellung eines Softwareprozesses. Softwareprozesse werden entweder als plangesteuert oder als agile Prozesse gesteuert.

**Plangesteuerte Modelle** sind Verfahren, bei denen alle Prozessaktivitäten im Voraus geplant werden und bei denen Fortschritt an diesem Plan gemessen wird.

In **agilen Vorgehensmodellen**, ist das Planen inkrementell (schrittweise) und es ist leichter, den Prozess zu verändern, um die Änderungen der Kundenanforderungen widerzuspiegeln.

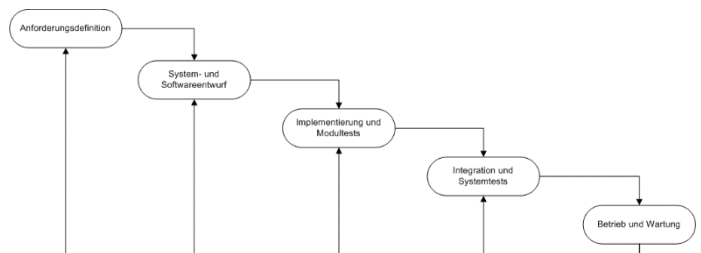
## PLANGESTEUERTE MODELLE

### Wasserfallmodell

Ein Modell, in dem es klar abgrenzte Entwicklungsphasen gibt. Grundsätzlich muss eine Phase erst vollständig abgeschlossen sein, bevor zur nächsten Phase fortgeschritten werden kann

Die wichtigsten Phasen des Wasserfallmodells spiegeln direkt die grundlegenden Entwicklungsaktivitäten wider:

- Analyse und Definition der Anforderungen
- System- und Softwareentwurf
- Implementierung und Modultest
- Integration und Systemtest
- Betrieb und Wartung



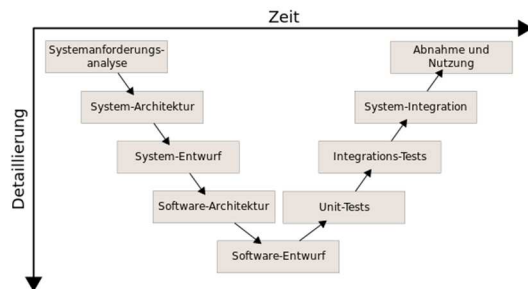
Im Prinzip gehen aus jeder Phase ein oder mehrere Dokumente hervor, die genehmigt oder abgenommen werden. Die nächste Phase sollte nicht beginnen, bevor nicht die vorherige abgeschlossen wurde. In der Praxis überlappen sich die Phasen und tauschen Informationen untereinander aus.

Das Wasserfallmodell ist konsistent zu anderen Systementwicklungsmodellen und erzeugt in jeder Phase Dokumentationen. Dadurch bleibt der Prozess durchschaubar, sodass Manager den Fortschritt gemäß Entwicklungsplan überwachen können.

Das Hauptproblem ist jedoch die starre Aufteilung des Projekts in verschiedene Phasen.

## **V-Modell**

Das V-Modell ist ein Vorgehensmodell in der Softwareentwicklung, bei dem der Softwareentwicklungsprozess in Phasen organisiert wird. Neben diesen Entwicklungsphasen definiert das V-Modell auch das Vorgehen zur Qualitätssicherung (Testen) phasenweise.



Es basiert auf dem Wasserfallmodell: Die Phasenergebnisse sind bindende Vorgaben für die nächsttiefere Projektphase. Der linke, nach unten führende Ast für die Spezifizierungsphasen schließt mit der Realisierungsphase ab. Eine Erweiterung gegenüber dem Wasserfallmodell sind die zeitlich nachfolgenden Testphasen, die im rechten, nach oben führenden Ast dargestellt werden. Den spezifizierenden Phasen stehen jeweils testende Phasen gegenüber, was in der Darstellung ein charakteristisches „V“ ergibt, das dem Modell auch den Namen gab. Diese Gegenüberstellung soll zu einer möglichst hohen Testabdeckung führen, weil die Spezifikationen der jeweiligen Entwicklungsstufen die Grundlage für die Tests (Testfälle) in den entsprechenden Teststufen sind.

## **Inkrementelle Entwicklung**

Ein Softwareentwicklungsansatz, bei dem die Software in Inkrementen geliefert und bereitgestellt wird.

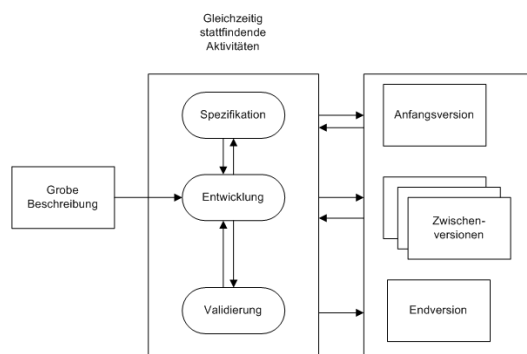
Dieser Ansatz verknüpft die Aktivitäten der Spezifikation, der Entwicklung und der Validierung. Das System wird als eine Folge von Versionen (Inkrementen) entwickelt, wobei jede Version neue Funktionalität zu der vorherigen hinzufügt.

Die inkrementelle Entwicklung basiert darauf, eine Anfangsimplementierung zu entwickeln, die Benutzer zu Kommentaren und Hinweisen zu dieser Implementierung aufzufordern und sie über mehrere Versionen hinweg zu verbessern, bis ein angemessenes System entstanden ist.

Die Spezifikation, die Entwicklung und die Validierung werden nicht als separate Abläufe betrachtet, sondern werden gleichzeitig ausgeführt, wobei sie untereinander Rückmeldungen zügig austauschen.

Inkrementelle Software-Entwicklung ist ein fundamentaler Teil des agilen Ansatzes; sie ist besser als ein Wasserfallansatz für die meisten Geschäftsfälle, E-Commerce und individuellen Systeme geeignet.

Jedes Inkrement oder jede Version des Systems besitzt einen Teil der Funktionalität, die vom Kunden gebraucht wird. In der Regel enthalten die frühen Systeminkremente die wichtigsten oder am dringendsten benötigten Funktionen. Dies bedeutet, dass der Kunde das System zu einem relativ frühen Zustand in der Entwicklung evaluieren kann um festzustellen, ob es den Anforderungen entspricht.

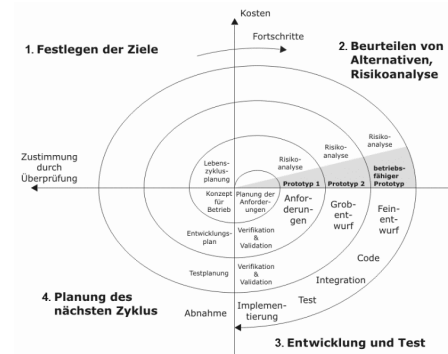


### Spiralmodell nach Boehm

Der Softwareprozess wird als Spirale dargestellt anstatt als eine Folge von Aktivitäten mit Rückwärtsbezügen von einer Aktivität zur anderen. Jede Windung der Spirale steht für eine Phase des Prozesses. So beschäftigt sich die innere Windung mit der Machbarkeit des Systems, die nächste mit der Definition der Systemanforderungen, die folgende mit dem Systementwurf, usw. Das Spiralmodell kombiniert die Vermeidung von Änderungen mit Änderungstoleranz. Es geht davon aus, dass Änderungen ein Ergebnis von Projektrisiken sind und beinhaltet explizite Risikomanagementaktivitäten, um diese Risiken zu reduzieren.

Jede Spirale ist in vier Segmente aufgeteilt:

- Ziele aufstellen
- Risiken einschätzen und verringern
- Entwicklung und Validierung
- Planung



Der Hauptunterschied zwischen dem Spiralmodell und anderen Vorgehensmodellen liegt in der ausdrücklichen Betrachtung der Risiken im Spiralmodell.

### Rational Unified Process (RUP)

Ein allgemeines Softwareprozessmodell, das Softwareentwicklung als iterative (schrittweise annähernde, wiederholende) Aktivität in vier Phasen darstellt, nämlich Konzeption, Entwurf, Konstruktion und Übergabe. Anders als beim Wasserfallmodell, bei dem die Phasen mit Prozessaktivitäten gleichgesetzt werden, sind die Phasen des RUP enger mit geschäftlichen als mit fachlichen Belangen verbunden.

Die Wiederholung innerhalb des RUP wird auf zwei Arten unterstützt. Jede Phase kann iterativ ausgeführt werden, wobei die Ergebnisse schrittweise entwickelt werden. Die Phasen können ebenfalls insgesamt iterativ ausgeführt werden.

Die statische Sicht des RUP konzentriert sich auf die Aktivitäten während des Entwicklungsprozesses. Diese werden in der RUP-Beschreibung als Arbeitsabläufe bezeichnet.

Der Prozess legt sechs Hauptarbeitsabläufe und drei unterstützende Arbeitsabläufe fest.

- Geschäftsprozessmodellierung
- Anforderungsanalyse
- Analyse und Entwurf
- Implementierung
- Tests
- Auslieferung
- Konfigurations- und Änderungsmanagement.
- Projektmanagement
- Infrastruktur

Der Vorteil von dynamischen und statischen Sichten liegt darin, dass die Phasen des Entwicklungsprozesses nicht mit speziellen Arbeitsabläufen verbunden sind. Zumindest in der Theorie können alle RUP-Arbeitsabläufe in allen Phasen des Prozesses aktiv sein.

## **AGILE METHODEN**

Softwareentwicklungsmethoden, die auf eine schnelle Softwarelieferung abzielen. Die Software wird in Inkrementen entwickelt und geliefert, wobei Prozessdokumentation und Bürokratie minimiert werden. Der Fokus der Entwicklung liegt auf dem Code selbst, weniger auf den unterstützenden Dokumenten.

### **Extreme Programming (XP)**

Extreme Programming ist wahrscheinlich die bekannteste und am häufigsten verwendete agile Methode. Dieser Ansatz wurde dadurch entwickelt, bekannte gute Praktiken wie iterative Entwicklung „ins Extreme“ zu steigern. Beim Extreme Programming werden Anforderungen in Form von Szenarios ausgedrückt, die User-Stories genannt werden. Diese Szenarios werden direkt in Form einer Abfolge von Aufgaben implementiert. Die Programmierer arbeiten paarweise zusammen, um Tests für die einzelnen Aufgaben zu entwickeln, bevor sie Code schreiben. Alle Tests müssen erfolgreich ausgeführt werden, wenn neuer Code in das System integriert wird. Zwischen den einzelnen Releases des Systems vergeht nur wenig Zeit.

Extreme Programming umfasst eine Reihe von Vorgehensweisen, welche die Prinzipien agiler Methoden widerspiegeln

- Die inkrementelle Entwicklung wird durch kleine häufige Releases des Systems erreicht
- Die Einbeziehung des Kunden wird durch die kontinuierliche Teilnahme eines Kundenbevollmächtigten an der Entwicklung erreicht.
- Durch Paarprogrammierung, kollektives Eigentum am Systemcode und einen geeigneten Entwicklungsprozesses, bei dem es keine extrem langen Arbeitszeiten gibt, werden Menschen statt Prozesse in den Vordergrund gerückt.
- Änderungen werden durch regelmäßige Systemreleases an die Kunden, Test-First-Entwicklung, Refactoring zur Vermeidung von Codedegeneration und stetige Integration von neuen Funktionalitäten eingebunden.
- Der Erhalt der Einfachheit wird durch ständiges Refactoring zur Verbesserung der Codequalität und die Verwendung einfacher Entwürfe erreicht, die zukünftigen Änderungen am System nicht unnötigen vorgreifen.

### **Vorgehensweise**

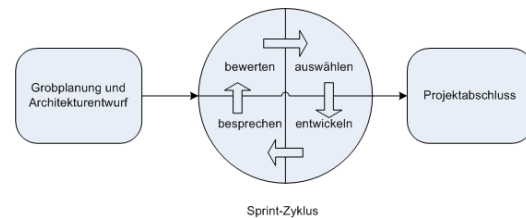
- Inkrementelle Planung (Anforderungen werden auf Story-Cards aufgezeichnet. Welche User-Stories in ein Release aufgenommen werden, wird aufgrund der verfügbaren Zeit und der jeweiligen Priorität entschieden. Die Entwickler teilen die User-Stories in Entwicklungsaufgaben auf)
- Kleine Releases
- Einfacher Entwurf
- Test-First-Entwicklung
- Refactoring (von allen Entwicklern wird erwartet, dass sie den Code einem kontinuierlichen Refactoring unterziehen, sobald Verbesserungsmöglichkeiten entdeckt werden)
- Paarprogrammierung
- Kollektives Eigentum (Entwickler arbeiten an allen Bereichen des Systems, keine Experten-Inseln)
- Kontinuierliche Integration
- Erträgliche Arbeitsgeschwindigkeit (große Mengen an Überstunden werden als nicht akzeptabel betrachtet)
- Kunde vor Ort (ein Bevollmächtigter des Kunden soll dem XP-Team ständig zur Verfügung stehen)

## Scrum

Scrum beruht auf sogenannten Sprints – kurze Entwicklung in Zyklen. Der Scrum-Ansatz ist eine Methode, die sich eher auf die Verwaltung iterativer Entwicklung als auf spezifische technische Aspekte des agilen Software-Engineerings konzentriert. Scrum schreibt nicht die Benutzung von bestimmten Programmiermodellen und testgetriebener Entwicklung vor.

Es gibt drei Phasen in Scrum:

- Allgemeine Planungsphase
- Serie von Sprint-Zyklen
- Projektabschluss



Das innovative an Scrum ist seine zentrale Phase, die Sprint-Zyklen.

Ein Sprint in Scrum ist eine Planungseinheit, in der die auszuführende Arbeit abgeschätzt wird, die zu entwickelnden Leistungsmerkmale ausgesucht werden und die Software implementiert wird. Am Ende eines Sprints wird die vollständige Funktionalität an die Projektbeteiligten ausgeliefert.

Die Hauptcharakteristika dieses Prozesses sind:

- Sprints haben eine feste Länge (normalerweise 2 – 4 Wochen)
- Ausgangsbasis für die Planung ist das Produkt-Backlog: eine Liste der Aufgaben, die erfüllt werden muss. Während der Bewertungsphase des Sprints wird diese Liste überprüft
- In der Auswahlphase sind alle Mitglieder des Projektteams einbezogen, um die Merkmale und Funktionalitäten auszuwählen, die entwickelt werden sollen
- Nachdem die Auswahl getroffen wurde, organisiert sich das Team zur Entwicklung der Software selbst. Es werden täglich kurze Treffen mit allen Teammitgliedern abgehalten. Während dieser Phase ist das Team vom Kunden und dem Unternehmen isoliert, die gesamte Kommunikation wird über den sogenannten Scrum-Master gelenkt. Die Rolle des Scrum-Masters ist es, das Entwicklerteam von äußeren Ablenkungen fernzuhalten.
- Am Ende des Sprints wird die Arbeit einer Besprechung unterzogen und den Projektbeteiligten vorgestellt. Der nächste Sprint-Zyklus beginnt.

Das Konzept von Scrum ist, dass das gesamte Team befugt ist, Entscheidungen zu fällen. Daher wurde der Begriff Projektmanager bewusst vermieden. Stattdessen fungiert der Scrum-Master als eine Art Moderator.

Scrum wurde ursprünglich für Teams entworfen, deren Mitglieder sich alle an einem Ort befinden und die jeden Tag für Meetings zusammenkommen können. Es wird derzeit versucht Scrum auf verteilte Entwicklungsumgebungen anzupassen.

## Fragen zu OO-Programmierung allgemein:

### Was ist der Unterschied zwischen einer Klasse und einem Objekt

Objekte werden durch Klassen erzeugt. Eine Klasse beschreibt eine bestimmte Art von Objekten; Objekte repräsentieren individuelle Instanzen einer Klasse.

Eine Klasse ist ein benutzerdefinierter Datentyp. Das Objekt ist ein Klassenwert.

#### Klasse

Eine Klasse definiert einen neuen Typ, beschreibt die Eigenschaften der Objekte und gibt somit den Bauplan an. Eine Klasse beschreibt, wie ein Objekt aussehen soll.

#### Objekt

Jedes Objekt ist ein Exemplar (auch Instanz oder Ausprägung genannt) einer Klasse

- Jedes Objekt hat eine Identität
- Jedes Objekt hat einen Zustand
- Jedes Objekt zeigt ein Verhalten

### Wie können Objekte erzeugt werden?

Objekte müssen in Java immer ausdrücklich erzeugt werden. Objekte werden mit dem Operator `new` erzeugt. Auf dieses Schlüsselwort folgen die Klasse des Objekts (also sein Typ) und eine optionale Argumenten-Liste in runden Klammern. Diese Argumente werden dem Konstruktor der Klasse übergeben, der die internen Felder des neuen Objekts initialisiert. Könnte die Speicherverwaltung freien Speicher reservieren und konnte der Konstruktor gültig durchlaufen werden, gibt der `new`-Ausdruck anschließend eine Referenz auf das frische Objekt an das Programm zurück.

Das Ergebnis des `new`-Operators ist eine Referenz auf das neue Objekt. Die Referenz wird in der Regel in eine Referenzvariable zwischengespeichert, um fortlaufende Eigenschaften vom Objekt nutzen zu können. Der Inhalt der Referenzvariable ist eine Adresse im Speicher, wo das Objekt gelagert ist.

Der Operator `new` erzeugt eine neue, aber uninitialisierte Instanz der Klasse. Dann wird die Konstruktormethode aufgerufen, der das neue Objekt implizit (in Form von `this`-Referenz) übergeben wird und der zusätzlich alle Argumente, die zwischen den runden Klammern stehen, explizit als Parameter übergeben werden. Der Konstruktor kann diese Argumente dann für die notwendigen Initialisierungsarbeiten verwenden.

Zum Erzeugen eines Objekts definiert man eine Referenzvariable, mit der ein neues Exemplar von einer Klasse (eine neue Instanz von dieser Klasse) referenziert wird.

New Operator -> Speicherplatzreservierung -> Übergabe der Parameter -> Konstruktoraufwurf

```
Adresse wifiAdresse = new Adresse("Wifi", 1180, "Wien");
```



**Was passiert bei der Vererbung?**

Schlüsselwort: `extends`

Vererbung erlaubt, eine Klasse als Erweiterung einer anderen Klasse zu definieren. Mit Vererbung kann das Problem der Code-Duplizierung gelöst werden. Statt zwei Klassen völlig unabhängig voneinander zu definieren, definiert man zuerst eine Klasse, die die Gemeinsamkeiten der beiden Klassen zusammenfasst. Bei der Vererbung, erbt eine Klasse (die Subklasse) die Eigenschaften und Methoden von einer anderen Klasse (die Superklasse).

```
public class Adresse extends JFrame {  
    // Programmcode  
}
```

**Was sind statische Methoden?**

Schlüsselwort: `static`

Der Modifikator `static` besagt, dass die Methode eine Klassenmethode ist. Klassenmethoden gehören zu einer Klasse und nicht zu einem Objekt. Wenn eine Klassenmethode von Code außerhalb der Klasse aufgerufen wird, muss sowohl der Name der Klasse als auch der Name der Methode angegeben werden. Klassenmethoden können das Schlüsselwort `this` nicht verwenden, da Klassenmethoden nicht zu einem Objekt gehören.

Eine Klassenmethode kann alle Klassenfelder (statische Variablen) und Klassenmethoden der eigenen Klasse (und aller anderen Klassen) verwenden, aber keine Instanzfelder oder Instanzmethoden, weil Klassenmethoden nicht zu einer bestimmten Instanz der Klasse gehören.

Definition

```
public class MyClass {  
    public static void myMethode() { // Programmcode }  
}
```

Aufruf

```
....  
    MyClass.myMethode();  
....
```

**Wofür steht „this“?**

Alle Instanzmethoden sind mit einem impliziten Parameter implementiert, der in der Methodensignatur nicht zu sehen ist. Dieses implizite Argument heißt `this` und enthält eine Referenz auf das Objekt, an dem die Methode aufgerufen worden ist. `This` steht für die eigene Objektinstanz.

Immer wenn eine Java-Methode auf die Felder ihrer Klasse zugreift, wird davon ausgegangen, dass dies ein Zugriff auf die Felder in dem Objekt ist, auf das der Parameter `this` verweist. Das gleiche gilt auch, wenn eine Instanzmethode eine andere Instanzmethode der gleichen Klasse aufruft.

Es gibt Fälle, in denen das Schlüsselwort `this` zwingend notwendig ist. Wenn beispielsweise ein Methodenparameter oder eine lokale Variable in einer Methode den gleichen Namen wie eines der Felder der Klasse hat, muss man `this` verwenden, um auf das Feld zuzugreifen, weil der Name alleine den Methodenparameter oder die lokale Variable bezeichnen würde.

Klassenmethoden können das Schlüsselwort `this` nicht verwenden, da Klassenmethoden nicht zu einem Objekt gehören.

```
public class Adresse {
    private String name;
    private int plz;
    private String ort;

    public Adresse(String name, int plz, String ort) {
        this.name = name;
        this.plz = plz;
        this.ort = ort;
    }
}
```

Es gibt noch einen speziellen Anwendungsfall für das Schlüsselwort `this`, wenn eine Klasse mehrere Konstruktoren hat: mit `this` kann ein Konstruktor von einem anderen Konstruktor der gleichen Klasse aufgerufen werden. Welcher Konstruktor aufgerufen wird, hängt von der Anzahl und den Typen der Argumente ab. Dieser Aufruf darf nur als erste Anweisung in einem Konstruktor stehen.

```
public class Adresse {
    private String name;
    private int plz;
    private String ort;

    // der grundlegende Konstruktor
    public Adresse(String name, int plz, String ort) {
        this.name = name;
        this.plz = plz;
        this.ort = ort;
    }

    // dieser Konstruktor verwendet this(), um obigen Konstruktor aufzurufen
    public Adresse() {
        this("Wien", 1180, "Wien");
    }
}
```

**Was versteht man unter Polymorphismus?**

Was: Vielgestaltigkeit

Wofür: Flexibilität

Von Polymorphie spricht man in Java beispielsweise, wenn zwei Klassen denselben Methodennamen verwenden, aber die Implementierung der Methoden sich unterscheidet. Häufig wird Polymorphie bei der Vererbung verwendet, d.h. dass einer Variable nicht nur Objekte der bei der Deklaration angegebenen Klasse annehmen kann, sondern auch Objekte der Kindsklassen. Dies funktioniert nur, weil jede Kindsklasse auch alle Methoden und Attribute ihrer Elternklassen implementieren muss.

Somit ist gewährleistet, dass alle Kindsklassen über dieselben Methoden verfügen wie die Elternklasse. Die Methoden können jedoch unterschiedlich implementiert werden, man spricht hier vom "Überschreiben" der Methode.

Die Polymorphie der objektorientierten Programmierung ist eine Eigenschaft, die immer im Zusammenhang mit Vererbung und Schnittstellen (Interfaces) auftritt. Eine Methode ist polymorph, wenn sie in verschiedenen Klassen die gleiche Signatur hat, jedoch erneut implementiert ist.

Gibt es in einem Vererbungsbaum einer Klassenhierarchie mehrere Methoden auf unterschiedlicher Hierarchieebene, jedoch mit gleicher Signatur, wird erst zur Laufzeit bestimmt, welche der Methoden für ein gegebenes Objekt verwendet wird (dynamisches Binden). Bei einer mehrstufigen Vererbung wird jene Methode verwendet, die direkt in der Objektklasse (d. h. jene Klasse, von der das Objekt ein Exemplar ist) definiert ist, oder jene, die im Vererbungsbaum am weitesten „unten“ liegt (d. h. die Methode, die von der Vererbung her am nächsten ist).

Late Binding

Nicht der Compiler entscheidet welche Methode (Super- oder Subklasse) verwendet wird, sondern erst die Laufzeit.

### Was sind abstrakte/virtuelle Klassen?

Schlüsselwort: `abstract`

Diese Klasse kann nicht instanziiert werden, und gilt nur als Grundlage für Subklassen.

Mit dem Modifikator `abstract` kann man in Java eine Methode definieren, ohne sie zu implementieren. Eine derart implementierte Methode hat keinen Rumpf; auf die Signaturdefinition folgt lediglich ein Semikolon.

- Eine Klasse mit einer abstrakten Methode ist automatisch auch selbst abstrakt und muss auch als solche deklariert werden
- Eine abstrakte Klasse kann nicht instanziiert werden
- Eine von der abstrakten Klasse abgeleitete Klasse kann nur dann instanziiert werden, wenn sie alle abstrakten Methoden ihrer Superklasse überschreibt und für jede eine Implementation (also einen Methodenrumpf) bereitstellt.
- Wenn eine von einer abstrakten Klasse abgeleitete Klasse nicht alle geerbten abstrakten Klassen implementiert, dann ist auch die abgeleitete Klasse abstrakt
- `static`-, `private`- und `final`-Methoden können nicht abstrakt sein, weil diese Methoden nicht von der abgeleiteten Klasse überschrieben werden können. Entsprechend kann auch eine als `final` deklarierte Klasse keine abstrakten Methoden enthalten
- eine Klasse kann auch dann als `abstract` deklariert werden, wenn sie keine abstrakten Methoden enthält. Damit zeigt man an, dass die Implementation auf irgendeine Art und Weise unvollständig ist und nur als Basisklasse für eine oder mehrere abgeleitete Klassen dienen soll. Solche Klassen können nicht instanziiert werden.
- Zugriff auf eine abstrakte Klasse ist nur durch Vererbung möglich.
- Es können auch konkrete Methoden enthalten sein (nicht jedoch beim Interface) -> gemischt abstraktes Werkzeug

### Was ist ein Konstruktor?

Konstruktoren ermöglichen, dass ein Objekt nach seiner Erzeugung in einen gültigen Zustand versetzt wird. Diese Aufgabe wird auch als Initialisierung bezeichnet. Ein Konstruktor initialisiert ein Objekt in einen vernünftigen Zustand.

Jede Klasse hat mindestens einen Konstruktor. Wenn kein Konstruktor explizit definiert wird, erzeugt Java einen Default-Konstruktor, der keine Argumente erwartet und keine besonderen Initialisierungen erwartet.

- Hat keinen Rückgabewert (nicht einmal `void`)
- Hat den Namen der Klasse
- Ziel ist es, Eigenschaften mit vernünftigen Startwerten zu definieren
- Wird nur ein Mal aufgerufen (bei der Objekterstellung)

```
public class Adresse {
    private int plz;

    // Konstruktor
    public Adresse(int plz) {
        this.plz = plz;
    }
}
```

## Was sind Exceptions?

Schlüsselwörter: `throws`, `try...catch...finally`

Eine Ausnahme ist in Java ein Objekt. Der Typ dieses Objektes ist `java.lang.Throwable` oder eine von `Throwable` abgeleitete Klasse. `Throwable` hat zwei standardmäßig abgeleitete Klassen: `java.lang.Error` und `java.lang.Exception`.

Ausnahmen, die von `Error` abgeleitet sind, weisen normalerweise auf nicht behebbare Probleme hin (VM hat keinen Speicher mehr, Klassendatei ist defekt oder kann nicht gelesen werden,...). Solche Ausnahmen werden normalerweise nicht abgefangen. Jedes Ausnahmeobjekt von der Klasse `Error` ist ungeprüft.

Ausnahmen, die von `Exception` abgeleitet sind, weisen auf weniger schwerwiegende Zustände hin (`ArrayIndexOutOfBounds`,...). Solche Ausnahmen können sinnvoll abgefangen und behandelt werden.

Das Wort `Exception` (Ausnahme) leitet sich von einer Ausnahmesituation her – also einer Situation, die normalerweise im Programm nicht auftaucht (Division durch Null,...). Um diese Fälle, haben die Entwickler von Java die Klasse `java.lang.Exception` entwickelt: Objekte, die zur Laufzeit generiert werden und einen Fehler anzeigen. Eine `Exception` ist ein Objekt, das Informationen über einen Programmfehler hält.

Wenn der Java-Interpreter eine `throw`-Anweisung ausführt, wird die normale Programmausführung unmittelbar angehalten und der Interpreter sucht nach einer Ausnahmebehandlungsroutine (`exception handler`), die die Ausnahme abfangen, also behandeln kann.

Einen überwachten Programmbereich (Block) leitet das Schlüsselwort `try` ein. Dem `try` folgt in der Regel ein `catch`-Block, in dem der Programmcode steht, der den Fehler behandelt.

```
try {  
    // Programmcode  
}  
catch(ArithmeticException e) {  
    // Programmcode  
}  
finally {  
    // Programmcode  
}
```

Java hat verschiedene Regeln zum Arbeiten mit geprüften und ungeprüften Ausnahmen. Wenn man eine Methode schreibt, die eine geprüfte Ausnahme auslöst, dann muss man eine `throws`-Klausel verwenden, um die Ausnahme im Methodenkopf zu deklarieren. Solche Ausnahmen werden geprüfte Ausnahmen genannt, weil der Java-Compiler prüft, ob man die Ausnahme in den Methodenköpfen deklariert hat und einen Fehler meldet, falls das nicht zutrifft.

Wenn eine Methode eine andere Methode aufruft, die eine geprüfte Ausnahme auslösen kann, dann muss man diese Ausnahme entweder in einer eigenen Ausnahmebehandlungsroutine behandeln oder mit `throws` deklarieren, um zu zeigen, dass die Methode die Ausnahme weiterreichen kann.

## Fragen zu Java:

### Was ist der wesentliche Unterschied zwischen Java und einer herkömmlichen Programmiersprache?

Java ist eine objektorientierte Programmiersprache, die sich durch einige zentrale Eigenschaften auszeichnet.

#### Objektorientierung in Java

Java ist als Sprache entworfen worden, die es einfach machen sollte, große, fehlerfreie Anwendungen zu schreiben. Objektorientierung versucht, die Komplexität des Software-Problems besser zu modellieren.

#### Bytecode

Im Gegensatz zu herkömmlichen Übersetzern einer Programmiersprache, die in der Regel Maschinencode für eine spezielle Plattform und einen bestimmten Prozessor generieren, erzeugt der Java-Compiler Programmcode, den sogenannten Bytecode, für eine virtuelle Maschine.

#### Plattformunabhängigkeit

Eine zentrale Eigenschaft von Java ist seine Plattformunabhängigkeit bzw. Betriebssystemunabhängigkeit. Dies wird durch zwei zentrale Konzepte erreicht. Zum einen bindet sich Java nicht an einen bestimmten Prozessor oder eine bestimmte Architektur, sondern der Compiler generiert Bytecode, den eine Laufzeitumgebung dann abarbeitet. Zum anderen abstrahiert Java von den Eigenschaften eines konkreten Betriebssystems, schafft etwa eine Schnittstelle zum Ein-/Ausgabesystem oder eine API für grafische Oberflächen. Die Java-Laufzeitumgebung ist Vermittler zwischen den Java-Programmen und der eigentlichen Betriebssystem-API.

#### Ausführung des Bytecodes durch eine virtuelle Maschine

Damit der Programmcode des virtuellen Prozessors ausgeführt werden kann, führt nach der Übersetzungsphase die Laufzeitumgebung, also die Java Virtual Machine (JVM), den Bytecode aus. Die Laufzeitumgebung lädt den Bytecode, prüft ihn und führt ihn in einer kontrollierten Umgebung aus.

#### Java als Sprache, Laufzeitumgebung und Standardbibliothek

Java ist nicht nur eine Programmiersprache, sondern ebenso ein Laufzeitsystem. Zu der Programmiersprache und der JVM kommt ein Satz von Standardbibliotheken für grafische Oberflächen, Ein-/Ausgabe und Netzwerkoperationen. Das bildet die Basis für höherwertige Dienste wie Datenbankverbindungen oder Web-Services.

#### Garbage-Collector

Die Java-Laufzeitumgebung kümmert sich selbständig um die Verwaltung von Objekten. Der GC ist ein Teil der Laufzeitumgebung von Java. Nach dem expliziten Generieren eines Objekts überwacht Java permanent, ob das Objekt noch gebraucht wird, also referenziert wird. Der GC ist ein nebenläufiger Thread im Hintergrund, der nicht referenzierte Objekte findet, markiert und dann von Zeit zu Zeit entfernt.

#### Ausnahmebehandlungen / Exceptions

Java unterstützt ein modernes System, um mit Laufzeitfehlern umzugehen. In die Programmiersprache wurden Ausnahmen (exceptions) eingeführt: Objekte, die zur Laufzeit generiert werden und einen Fehler anzeigen. Diese Problemstellen können durch Programm-Konstrukte gekapselt werden.

**Welche 8 elementaren Typen gibt es in Java?**

- `boolean` (true oder false)
- `char` (16-Bit-Unicode-Zeichen)
- `byte` (Ganzzahlen)
- `short` (Ganzzahlen)
- `int` (Ganzzahlen)
- `long` (Fließkommazahlen)
- `float` (Fließkommazahlen)
- `double` (Fließkommazahlen)

**Was versteht man unter Casting?**

Typanpassung

Automatische (implizite) Typanpassung (vergrößernde Konvertierung)

Daten eines kleineren Datentyps werden automatisch dem größeren angepasst. Der Compiler nimmt diese Anpassung selbständig vor. Z.B. werden Werte der Datentypen `byte` und `short` bei Rechenoperationen automatisch in `int` umgewandelt. Ist ein Operand `long`, dann werden alle Operanden auf `long` erweitert.

Explizite Typanpassung (verkleinernde Konvertierung)

Der größere Datentyp kann einem kleineren Typ mit möglichem Verlust von Information zugewiesen werden. Ein Cast wird durchgeführt, wenn man den gewünschten Typ, umgeben von einer runden Klammer vor den zu konvertierenden Wert schreibt.

Umwandlung einer Fließkommazahl in eine Ganzzahl:

```
int n = (int) 3.1415;    //n =3
```

**Wie werden in Java Parameter übergeben?**

Generell gilt, dass beim Methodenaufzuruf der Wert der Variablen, nicht deren Speicherort (Referenz) übergeben wird ('call by value'). Die Übergabe von primitiven und Referenztypen muss gesondert betrachtet werden.

Übergabe primitiver Werte

Wert der Variable wird kopiert. Beim Aufruf einer Methode wird ein formaler Parameter vor der Übergabe an die Methode selbst validiert und dann das Ergebnis als tatsächlicher Parameter an die Methode weitergeleitet. Die dortige Verwendung hat keinen Einfluss auf den Wert des formalen Parameters.

Übergabe von Referenzwerten

Wert der Variable (in diesem Fall die Speicheradresse) wird kopiert. Diese Änderung ist dauerhaft, bezieht sich also nicht nur auf den tatsächlichen Parameter. Das anschließende Setzen des Parameters auf null zeigt, dass hier tatsächlich der Wert, nicht die Referenz selbst geändert wurde

**Was ist ein Interface und wozu werden Interfaces verwendet?**

Schlüsselwort: `interface`

Nicht immer soll eine Klasse sofort ausprogrammiert werden, zum Beispiel dann nicht, wenn die Oberklasse lediglich Methoden für die Unterklasse vorgeben möchte, aber nicht weiß, wie sie diese implementieren soll. Da Java nur Einfachvererbung kennt, ist es schwierig, Klassen mehrere Typen zu geben. Da es aber möglich sein soll, dass in der objektorientierten Modellierung eine Klasse mehrere Typen in einem Schritt zu besitzen, gibt es das Konzept der Schnittstelle (Interface). Eine Klasse kann dann neben der Oberklasse eine beliebige Anzahl Schnittstellen implementieren und auf diese Weise weitere Typen sammeln.

Schnittstellen sind eine gute Ergänzung zu abstrakten Klassen/Methoden. Denn im objektorientierten Design wollen wir das Was vom Wie trennen. Abstrakte Methoden sagen wie Schnittstellen etwas über das Was aus, aber erst die konkrete Implementierung realisiert das Wie.

Vorteil: bestimmte Funktionen heißen überall gleich, haben die gleichen Parameter und gleichen Rückgabewerte -> Vereinheitlichung

Es dürfen nur konstante Eigenschaften und abstrakte Methoden enthalten sein. Von Schnittstellen können keine Exemplare gebildet werden und der Versuch einer Objekterzeugung führt zu einem Compilerfehler.

Eine Schnittstelle enthält keine Implementierungen, sondern deklariert nur den Kopf einer Methode.

```
Interface Buyable {  
    double price();  
}
```

Das Definieren eines Interface erfolgt fast genauso wie das Definieren einer abstrakten Klasse, es werden lediglich die Schlüsselwörter `abstract` und `class` durch das Schlüsselwort `interface` ersetzt.

Wenn ein Interface definiert wird, wird ein neuer Referenztyp, genau wie beim Definieren einer Klasse erzeugt. Wie der Name schon andeutet, spezifiziert ein Interface eine Schnittstelle (eine API) für eine bestimmte Funktionalität. Es definiert aber keinerlei Implementierung dieser Schnittstelle.

- Ein Interface enthält keinerlei Implementierung. Alle Methoden eines Interfaces sind implizit abstrakt, auch wenn der Modifikator `abstract` nicht angegeben wird. Interface-Methoden haben keine Implementierung, ein Semikolon ersetzt den Methodenrumpf. Weil Interfaces nur abstrakte Methoden enthalten können und Klassenmethoden (`static`) wiederum nicht abstrakt sein können, müssen alle Methoden eines Interfaces Instanzmethoden sein.
- Ein Interface definiert eine öffentliche API. Alle Methoden eines Interface sind implizit `public`, auch wenn dieser Modifikator nicht angegeben wird. Es ist ein Fehler, eine Methode eines Interfaces als `protected` oder `private` zu definieren.
- Obwohl Klassen Daten und Methoden definieren, die auf diesen Daten operieren, kann ein Interface keine Instanzfelder definieren. Felder sind ein Implementationsdetail, und ein Interface ist eine reine Spezifikation ohne jede Implementation. Die einzigen Felder, die in einem Interface erlaubt sind, sind Konstanten, die sowohl als `static` als auch `final` deklariert sind.
- Ein Interface kann nicht instanziiert werden und definiert deswegen auch keinen Konstruktor.

Implementieren von Schnittstellen – Schlüsselwort: `implements`

```
public Class MyClass implements BuyAble {  
  
    double price() {  
        int n = 100;  
    }  
}
```



**Welche Sichtbarkeitsoperatoren gibt es in Java und was bedeuten sie?**Public (öffentlich)

Der Sichtbarkeits-Modifizierer `public` an Klassen, Konstruktoren, Methoden und sonstigen Klassen-Innereien bestimmt, dass all diese markierten Elemente von außen für jeden sichtbar sind. Es spielt dabei keine Rolle, ob sich der Nutzer im gleichen oder in einem anderen Paket befindet.

Private (privat)

Der Sichtbarkeits-Modifizierer `private` verbietet allen von außen zugreifenden Klassen den Zugriff auf Eigenschaften. Die mit `private` deklarierten Methoden und Variablen sind nur innerhalb der eigenen Klasse sichtbar. Wenn die Klasse erweitert wird, sind die privaten Elemente für Unterklassen nicht sichtbar.

Ohne Modifizierer (paketsichtbar)

Steht kein ausdrücklicher Sichtbarkeits-Modifizierer an den Eigenschaften, gilt die Paketsichtbarkeit. Sie sagt aus, dass die paketsichtbaren Klassen nur von anderen Klassen im gleichen Paket gesehen werden können. Für die Eigenschaften gilt das Gleiche.

Protected (geschützt)

`Protected` umfasst zwei Eigenschaften

- `protected`-Eigenschaften werden an alle Unterklassen vererbt
- Klassen, die sich im gleichen Paket befinden, können alle `protected`-Eigenschaften sehen, denn `protected` ist eine Erweiterung der Paketsichtbarkeit

Bei einer Klasse ist nur `public` oder `default` (package scope) erlaubt

**Was versteht man unter Garbage Collection?**

Automatische Speicherbereinigung.

Der Garbage Collector ist ein Teil der Laufzeitumgebung von Java und kümmert sich selbständig um die Verwaltung von Objekten. Der Garbage Collector ist ein automatisch ablaufender Prozess, der dafür zuständig ist, alle zur Laufzeit eines Programms existierende Objekte regelmäßig daraufhin zu überprüfen, ob sie noch referenziert (gebraucht) werden. Gibt es auf ein Objekt keine Referenz mehr, kann das Objekt gelöscht werden und der von ihm belegte Speicherplatz freigegeben werden.

**Was sind Generics und wozu werden sie verwendet?**

Generics gewährleisten bessere Typsicherheit, da nur spezielle Objekte in die Datenstruktur kommen. Mit den Generics lassen sich bei der Konstruktion einer Collection-Datenstruktur angeben, welche Typen zum Beispiel in der Datenstruktur-Liste erlaubt sind.

Die Schreibweise `List<String>` deklariert eine Liste, die Strings enthält.

Der Begriff steht synonym für „parametrisierte Typen“. Die Idee dahinter ist, zusätzliche Variablen für Typen einzuführen. Diese Typ-Variablen repräsentieren zum Zeitpunkt der Implementierung unbekannte Typen. Erst bei der Verwendung der Klassen, Schnittstellen und Methoden werden diese Typ-Variablen durch konkrete Typen ersetzt. Damit kann typsichere Programmierung gewährleistet werden.

Klasse, die andere Objekte enthält, der Datentyp wird in der Klassendefinition allgemein gehalten und erst wenn ein konkretes Objekt angelegt wird, wird der Typ festgelegt (Typsicherheit).

**Wie funktioniert in Java die Event-Programmierung (z.B. in Swing)?**

Beim Arbeiten mit grafischen Oberflächen interagiert der Benutzer mit Komponenten. Er bewegt die Maus im Fenster, klickt eine Schaltfläche an oder verschiebt einen Rollbalken. Das grafische System beobachtet diese Aktionen des Benutzers und informiert die Applikation über die anfallenden Ereignisse. Dann kann das laufende Programm entsprechend reagieren.

Im Ereignismodell von Java gibt es eine Reihe von Ereignisauslösern (Event Sources), wie zum Beispiel Schaltflächen oder Schieberegler. Die Ereignisse können vom Benutzer der grafischen Oberfläche kommen, aber auch auf eigene Auslöser zurückzuführen sein, zum Beispiel wenn im Hintergrund eine Tabelle geladen wird.

Neben den Ereignisauslösern gibt es eine Reihe von Interessenten (Listener), die gerne informiert werden wollen, wenn ein Ereignis aufgetreten ist. Da der Listener in der Regel nicht an allen ausgelösten Oberflächen-Events interessiert ist, sagt er einfach, welche Ereignisse er empfangen möchte. Dies funktioniert so, dass er sich bei einer Ereignisquelle anmeldet, und diese informiert ihn, wenn sie ein Ereignis aussendet. Auf diese Weise leidet die Systemeffizienz nicht, da nur diejenigen informiert werden, die auch Verwendung für das Ereignis haben. Für jedes Ereignis gibt es einen eigenen Listener, an den das Ereignis weitergeleitet wird.

Einige Listener und welche Ereignisse sie melden

<code>ActionListener</code>	Der Benutzer aktiviert eine Schaltfläche bzw. ein Menü oder drückt die Enter-Taste auf einem Textfeld
<code>WindowListener</code>	Der Benutzer schließt ein Fenster oder möchte es verkleinern
<code>MouseListener</code>	Der Benutzer drückt eine Maustaste
<code>MouseMotionListener</code>	Der Benutzer bewegt die Maus

Dem Listener übergibt das Swing-Grafiksystem jeweils ein Ereignis-Objekt, etwa dem `ActionListener` ein `ActionEvent`-Objekt, dem `WindowListener` ein `WindowEvent`-Objekt usw. Die Einzigen, die etwas aus der Reihe tanzen, sind `MouseListener` und `MouseMotionListener`, denn beide melden `MouseEvent`-Objekte.

Damit die Quelle Ereignisse sendet und der Client darauf reagiert, sind zwei Schritte durchzuführen:

1. Implementierung des Listeners
2. Anmelden des Listeners

```
myframe.addWindowListener( new DialogWindowClosingListener() );
```

### Listener implementieren

Der Listener selbst ist eine Schnittstelle, die von den Interessenten implementiert wird. Da die Ereignis-Schnittstelle Callback-Methoden vorschreibt, muss der Interessent diese Operation implementieren. Wird im nächsten Schritt ein Listener mit dem Ereignisauslöser verbunden, kann diese Ereignisquelle davon ausgehen, dass der Listener die entsprechende Methode besitzt. Diese ruft die Ereignisquelle bei einem Ereignis später auf.

### Listener beim Ereignisauslöser anmelden/abmelden

Hat der Listener die Schnittstelle implementiert, wird er mit dem Ereignisauslöser verbunden.

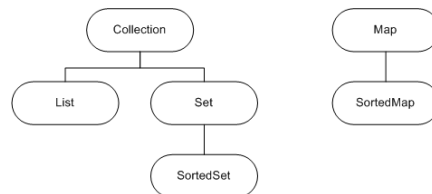
```
addEreignisListener(EreignisListener)  
removeEreignisListener(EreignisListener)
```

```
public class BookGUI extends JFrame {  
    ActionListener myActionListener = new BookGUIActionListener(this,loginUser);  
    searchButton.addActionListener(myActionListener);  
    ....  
}
```

```
public class BookGUIActionListener implements ActionListener {  
    @Override  
    public void actionPerformed(ActionEvent event) {  
        if (event.getActionCommand().contains("alle anzeigen")) {  
            guiBook.getSearchText().setText("");  
            guiBook.createBookTable();  
        }  
        ....  
}
```

**Geben Sie einen Überblick über das Collection Framework!**

Eine Sammlung von Interfaces, die die Organisation von Objekten in „Container“ unterstützt. Das Container-Framework von Java (Java Collection Framework) ist eine Menge von wichtigen Hilfsklassen und Interfaces im Package `java.util`, dass mit Collections von Objekten arbeitet. Es gibt zwei grundlegende Typen von Container. Eine `Collection` ist eine Gruppe von Objekten, während eine `Map` eine Menge von Abbildungen oder Assoziationen zwischen zwei Objekten ist. `SortedSet` und `SortedMap` sind spezialisierte Mengen und Abbildungen, die ihre Elemente sortiert aufbewahren. Allesamt sind Interfaces.

**java.util.Collection**

Interface, um eine Gruppe von Objekten zu organisieren, Basis-Definitionen für Hinzufügen und Entfernen von Objekten

`java.util.List`

Collection Interface, das zusätzlich jedem Element eine fixe Position zuweist (indiziert)

- `ArrayList` (Listenfunktionen Abbildung auf ein Feld)
- `LinkedList` (Referenz auf den jeweiligen Vorgänger und Nachfolger)

`java.util.Set`

Collection Interface, das keine doppelten Elemente erlaubt (Mengen)

- `HashSet` (schnelles Hash-Verfahren)
- `TreeSet` (Baumstruktur)
- `LinkedHashSet` (schnelle Mengenimplementierung, die sich parallel auch die Reihenfolge der Elemente merkt)

**java.util.Map (Key-Value Paare)**

Interface, das die Zuordnung von Elementen zu sogenannten Schlüsseln unterstützt; erlaubt, Elemente mit dem zugehörigen Schlüssel anzusprechen; `Map` ist keine Unterklasse von `Collection`

- `TreeSet` (Baumstruktur)
- `TreeMap` (Binärbaum Sortierung)
- `LinkedHashMap` (schneller Assoziativspeicher, der sich parallel auch die Reihenfolge der Elemente merkt)

`java.util.Iterator`

Ein Iterator ermöglicht es, alle Elemente eines `Collection`-Objekts kontrolliert (Element für Element) zu durchlaufen und abzuarbeiten.

`java.util.Queue`

Deklariert Operationen für alle Warteschlangen

**Geben Sie einen Überblick über die Stream-Klassen!**

Streams werden im Deutschen normalerweise als Datenströme bezeichnet. Streams werden verwendet, um Daten in ein Programm einzulesen bzw. Daten aus einem Programm auszugeben. Wann immer ein Programm mit seiner Umgebung kommunizieren möchte, kann dies über Streams geschehen. Stellen Sie sich Streams einfach als Rohre vor, die von einem Programm in die Programmumgebung führen und über die Daten zwischen dem Programm und der Programmumgebung ausgetauscht werden können. Die Standardbibliothek in Java bietet zahlreiche Rohre an, die in Form von Klassen im Paket `java.io` definiert sind. Der Paketname `io` steht für Input/Output, also die Datenein- und -ausgabe.

Beachten Sie, dass Benutzerschnittstellen, die mit AWT oder Swing entwickelt sind, zwar auch eine Form der Datenein- und -ausgabe ermöglichen, diese jedoch nicht zu den Streams gezählt wird. Während AWT und Swing das Augenmerk auf eine übersichtliche Präsentation von Daten innerhalb der Programmoberfläche legen, sind Streams grundlegendere Mechanismen zur Datenein- und -ausgabe, die in keiner Weise mit AWT und Swing vergleichbar sind und auch nicht in Konkurrenz zu diesen stehen.

Java unterscheidet zeichen- und byteorientierte Streams.

Zeichenorientierte Streams werden verwendet, wenn Buchstaben, Wörter oder Texte in irgendeiner Form zwischen einem Programm und seiner Umgebung ausgetauscht werden müssen.

Byteorientierte Streams werden für alle anderen Arten von Datenaustauschvorgängen verwendet, also wenn zum Beispiel Grafiken oder Objekte ausgetauscht werden sollen. Der Grund für diese Unterscheidung ist, dass Zeichen in Java auf 2 Bytes basieren, Rohdaten, wie sie für sämtliche anderen Datenformate verwendet werden, jedoch jeweils auf 1 Byte.

Die Unterscheidung in zeichen- und byteorientierte Streams wird anhand der Klassen in der Standardbibliothek deutlich.

Zeichenorientierte Streams sind in Form von Klassen implementiert, die alle entweder von der Klasse `java.io.Reader` oder von der Klasse `java.io.Writer` abgeleitet sind.

Alle Klassen, die byteorientierte Streams implementieren, sind Kindklassen von `java.io.InputStream` oder von `java.io.OutputStream`.

Streams können nicht nur nach Zeichen- oder Byteorientierung in zwei Gruppen eingeteilt werden, sondern sie unterscheiden sich auch danach, ob sie Daten in ein Programm einlesen oder Daten aus einem Programm ausgeben. Alle Klassen, die von `java.io.Reader` und `java.io.InputStream` abgeleitet sind, lesen Daten in ein Programm ein - einmal zeichenorientiert, einmal byteorientiert. Alle Klassen, die von `java.io.Writer` und `java.io.OutputStream` abgeleitet sind, geben Daten aus einem Programm aus - wiederum einmal zeichenorientiert, einmal byteorientiert.

Selbstverständlich gibt es die üblichen Ausnahmen, nämlich Streams, die sowohl Daten einlesen als auch ausgeben können. So gibt es zum Beispiel seit der allerersten Java-Version 1.0 eine Klasse `java.io.RandomAccessFile`, die Daten aus Dateien lesen als auch in Dateien speichern kann. In diesem Fall würden also nicht zwei Streams benötigt werden, die getrennt Daten lesen und schreiben, sondern es würde ein Objekt vom Typ `java.io.RandomAccessFile` reichen.

### Zeichenorientierte Streams

Ein- und Ausgabe von Buchstaben, Wörtern und Texten

Im Folgenden werden Ihnen einige Stream-Klassen vorgestellt, mit denen Daten zeichenorientiert in Java gelesen und ausgegeben werden können. Folgende erste Übersicht enthält einige Klassen, die von `java.io.Reader` abgeleitet sind und daher ein zeichenweises Einlesen von Daten in ein Programm ermöglichen.

- `java.io.BufferedReader` stellt eine gepufferte und daher effizientere Eingabemöglichkeit zur Verfügung. Dieser Stream dient also lediglich einer optimierten Eingabe.
- `java.io.InputStreamReader` liest Zeichen ein, die auf 1 Byte basieren, und wandelt sie in Zeichen um, die wie in Java üblich auf 2 Bytes basieren.
- `java.io.FileReader` ist von `java.io.InputStreamReader` abgeleitet und wird verwendet, um Zeichen aus Dateien in ein Programm einzulesen. Da Textdateien normalerweise so gespeichert sind, dass Zeichen auf 1 Byte basieren, wandelt diese Klasse automatisch die aus der Datei gelesenen Zeichen so um, dass sie wie in Java üblich auf 2 Bytes basieren.
- `java.io.StringReader` ermöglicht es, eine Zeichenkette vom Typ `java.lang.String` als Datenquelle zu nutzen. Es werden keine Daten aus der Programmumgebung gelesen, sondern ein herkömmlicher String als Datenquelle betrachtet.

Die folgende Übersicht stellt Ihnen einige Klassen vor, die von `java.io.Writer` abgeleitet sind und Gegenstücke zu den eben vorgestellten Klassen darstellen. Mit den folgenden Klassen können Zeichen von einem Programm geschrieben werden.

- `java.io.BufferedWriter` verwenden Sie, um Zeichen gepuffert auszugeben. Der einzige Sinn und Zweck dieser Klasse und der Pufferung ist, dass auf diese Weise eine Datenausgabe effizienter stattfindet.
- `java.io.OutputStreamWriter` wandelt Zeichen, die auf 2 Bytes basieren, in Zeichen um, die auf 1 Byte basieren.
- `java.io.FileWriter` wird verwendet, um Zeichen in eine Datei zu schreiben. Diese Klasse ist von `java.io.OutputStreamWriter` abgeleitet, so dass automatisch eine Umwandlung von Zeichen in Java, die auf 2 Bytes basieren, in Zeichen stattfindet, die auf 1 Byte basieren.
- `java.io.PrintWriter` bietet zahlreiche Methoden an, um Informationen verschiedenster Datentypen auszugeben. Diese Klasse vereinfacht letztendlich die Datenausgabe.
- `java.io.StringWriter` macht es möglich, Zeichen in ein Objekt vom Typ `java.lang.String` zu schreiben. Es werden keine Zeichen in die Programmumgebung geschrieben, sondern ein Objekt im Programm wird als Datensinke verwendet.

### Byteorientierte Streams

#### Ein- und Ausgabe beliebiger Daten

Nachdem Ihnen die Klassen für die zeichenorientierte Ein- und Ausgabe vorgestellt wurden, erhalten Sie im Folgenden einen Überblick über die byteorientierten Streams. Die folgende Übersicht enthält einige Klassen, um Daten byteorientiert in ein Programm einzulesen. Folgende Klassen sind alle von `java.io.InputStream` abgeleitet.

- `java.io.BufferedInputStream` verwenden Sie zur Performance-Steigerung. Diese Klasse stellt einen Puffer dar, über den Daten effizienter eingelesen werden können.
- `java.io.ByteArrayInputStream` stellt ein Array vom Typ `byte` dar, das als Datenquelle verwendet werden kann. Der Datenaustausch findet also bei diesem Stream nicht mit der Programmumgebung statt, sondern mit einem Objekt dieser Klasse, das direkt im Programm zur Verfügung steht.
- `java.io.FileInputStream` ermöglicht es, Daten aus Dateien zu lesen.
- `java.io.FilterInputStream` macht es möglich, Daten beim Einlesen aus einer Datenquelle zu filtern oder in einer bestimmten Weise zu verarbeiten.

Die folgende Übersicht stellt Ihnen abschließend die Klassen vor, die von `java.io.OutputStream` abgeleitet sind. Diese Klassen ermöglichen ein byteorientiertes Schreiben von Daten.

- `java.io.BufferedOutputStream` existiert allein aus Performance-Gründen. Mit Hilfe dieser Klasse finden Schreibvorgänge von Daten gepuffert und damit effizienter statt.
- `java.io.ByteArrayOutputStream` ermöglicht es Ihnen, ein Array vom Typ `byte` als Datensenke zu verwenden. Dieser Stream führt also nicht in die Programmumgebung, sondern Daten, die in diesen Stream geschrieben werden, werden in einem Array innerhalb des Programms gespeichert.
- `java.io.FilterOutputStream` gestattet eine Filterung oder Verarbeitung von Daten während des Schreibvorgangs.
- `java.io.FileOutputStream` verwenden Sie, wenn Sie Daten in eine Datei hineinschreiben möchten.
- `java.io.PrintStream` vereinfacht viele Schreibvorgänge, da diese Klasse zahlreiche Methoden anbietet, um Informationen verschiedenster Datentypen auszugeben.

**Was ist die JDBC?**

JDBC ist die inoffizielle Abkürzung für **J**ava **D**atabase **C**onnectivity und bezeichnet einen Satz von Schnittstellen, um relationale Datenbanksystem von Java zu nutzen. JDBC bietet ein standardisiertes Interface, um auf Datenbanken zuzugreifen.

Dem Programmierer gibt JDBC Methoden, um Verbindungen zu Datenbanken aufzubauen, Datensätze zu lesen oder neue Datensätze zu verfassen. Zusätzlich können Tabellen aktualisiert und Prozeduren auf der Serverseite ausgeführt werden.

1. Treiber laden, Registrierung beim DriverManager
2. URL definieren (Ziel-DB)
3. Verbindung zur DB herstellen
4. Statement-Objekt besorgen
5. Anfrage in SQL
6. Ergebnisse anzeigen

Programmierer müssen JDBC exakt implementieren. Datenbankhersteller ist für den Driver verantwortlich. Connection-String ist immer auf die Datenbank spezialisiert.

- JDBC – ODBC (Übersetzung von JDBC in ODBC)
- Verwendung von Zugriffen der Datenbank um direkt auf die Datenbank zugreifen zu können
- Middleware (stellt Zugriffe zur Verfügung)
- Java DB (Derby Driver)