















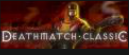






Anwendungsbeispiel: Architekturentwurf

← → STORE LIBRARY		STEAMOS + LINUX		VIEW	
GAMES		★	☁	STATUS	METASCORE
					LAST PLAYED
	Amnesia: The Dark Descent			Not installed	85
	Antichamber			Not installed	82
	Aquaria			Not installed	82
	Audiosurf 2		☁	Not installed	76
	The Banner Saga		☁	Not installed	80
	Bastion		☁	Not installed	86
	BioShock Infinite		☁	Not installed	94
	Borderlands 2		☁	Not installed	89
	Braid		☁	Not installed	90
	Broforce		☁	Not installed	83
	Chaos on Deponia		☁	Not installed	78
	Counter-Strike			Not installed	88
	Counter-Strike: Global Offensive		☁	Not installed	83
	Counter-Strike: Source		☁	Not installed	88
	Darkest Dungeon		☁	Not installed	84
	Day of Defeat			Not installed	79
	Dead Island		☁	Not installed	80
	Deathmatch Classic			Not installed	New
	Deponia		☁	Not installed	74

Erstellen eines Architekturentwurfs

Ausgangslage

Aufbau und Funktion eines Lastenhefts

1. 2. 3. 4. 5. 6. Qualitätsanforderungen:

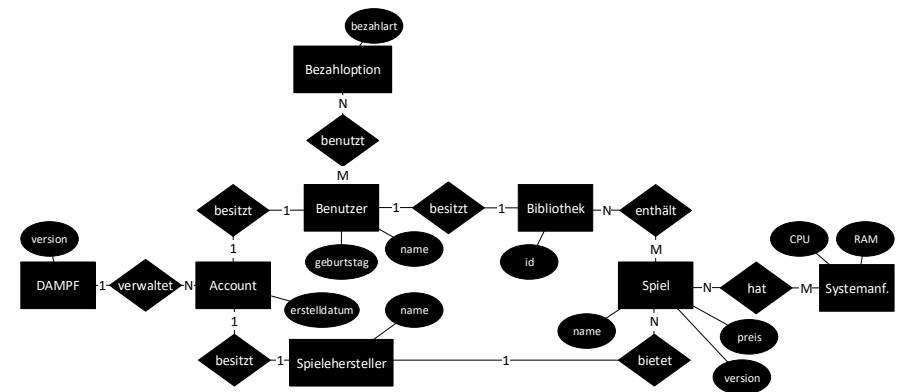
Produktqualität	sehr gut	gut	normal	irrelevant
Funktionalität		x		
Zuverlässigkeit	x			
Benutzbarkeit	x	x		
Effizienz			x	
Änderbarkeit			x	
Portierbarkeit				x

Die Benutzbarkeit der Funktionen mit dem die Benutzer arbeiten müssen sehr gut sein, da ein breites Spektrum an Nutzern angesprochen wird. Die Benutzbarkeit aller übrigen Funktionen muss gut sein, da sie von einem kleineren Klientel bedient wird (Spielehersteller und VAULT-Mitarbeiter).

7. Ergänzungen (wie z.B. Abgrenzungskriterien):
Buchhaltungsfunktionen (z.B. Erstellung von Abrechnungen zwischen VAULT und Spieleherstellern) gehören nicht zum Leistungsumfang.

Lasten- und Pflichtenheft 13

Requirements (Lastenheft)

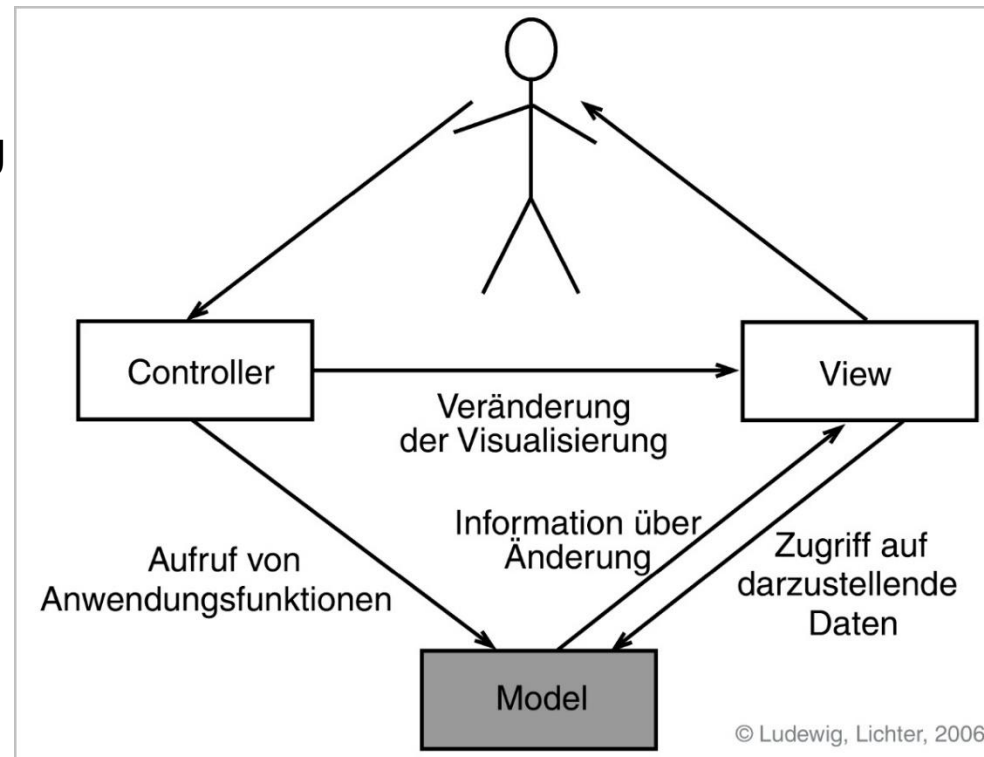


Domänenmodell

Architekturmuster – Model-View-Controller (MVC)

■ MVC Komponenten

- **Model** realisiert die fachliche Funktionalität der Anwendung (und kapselt die Daten der Anwendung).
- **View** präsentiert dem Benutzer die Daten des Models.
- Jeder View ist ein **Controller** zugeordnet. Dieser empfängt die Eingaben des Benutzers und reagiert darauf.



MVC in der Praxis

- Je nach benutztem Framework/Bibliothek ist die Implementierung unterschiedlich
- Konzept bleibt aber gleich!
- Im folgendem werden wir aus Übersichtlichkeit nur auf das Model-Layer eingehen



Entwurfsdokumentation

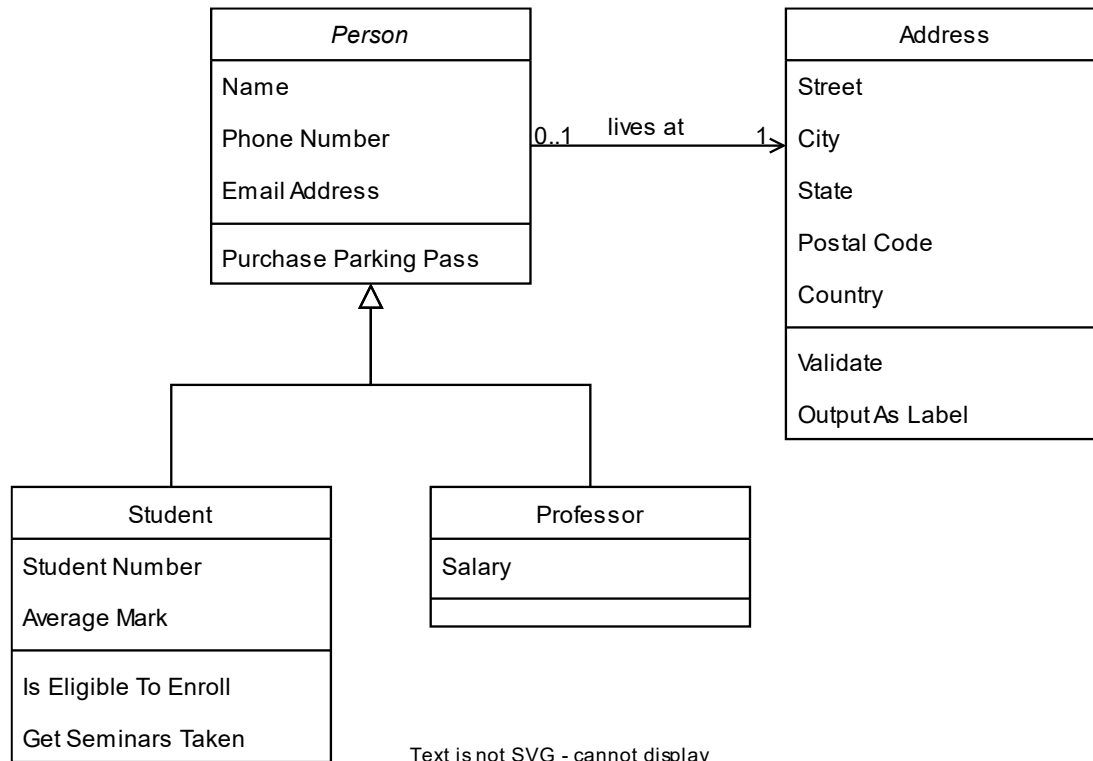
Dokumentation des Entwurfs

- Strukturelle Beschreibung
 - Repräsentiert strukturelles Design
 - Beschreibt große Komponenten und wie sie verbunden sind
 - Statisch
 - Beispiel: UML-Klassendiagramme
- Verhaltensbeschreibung
 - Beschreibt Verhalten des Systems und der Komponenten
 - Besonders nützlich in der Implementierung
 - Dynamisch
 - Beispiel: UML-Sequenzdiagramme



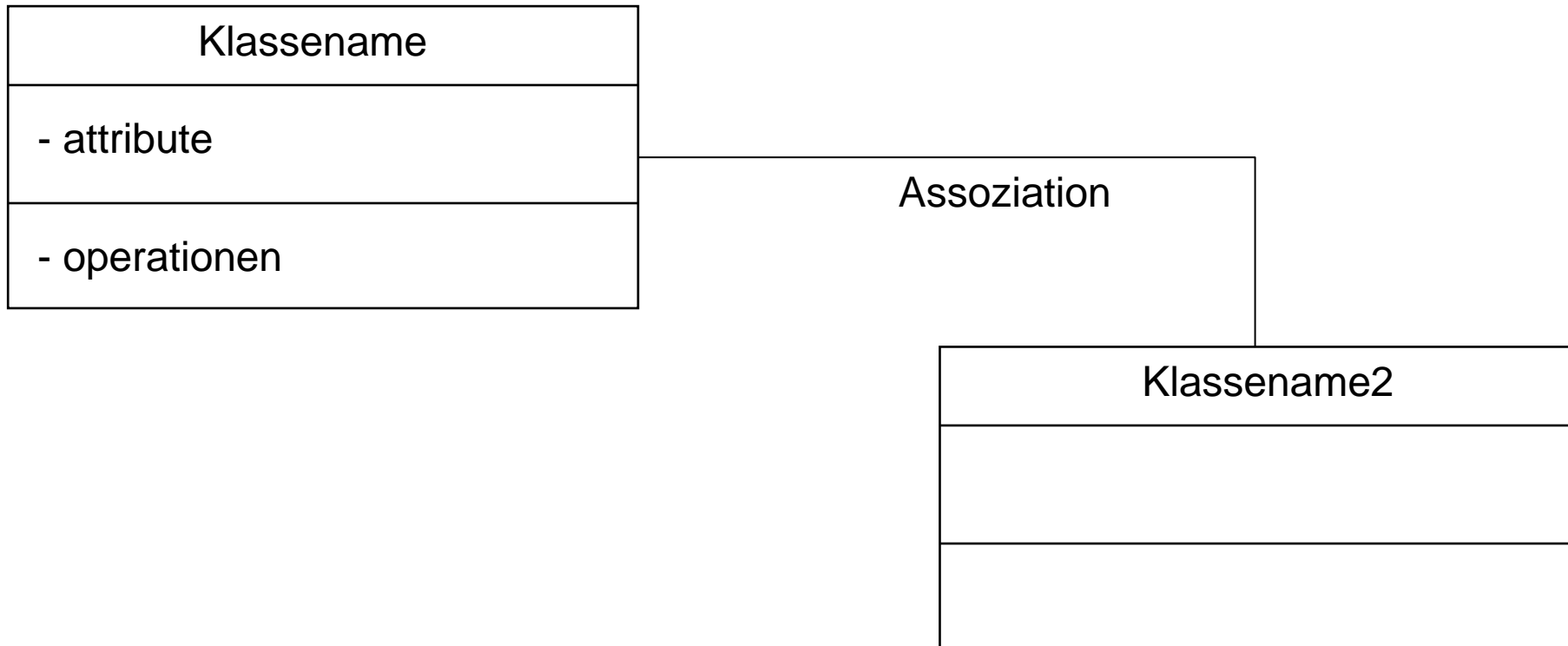
Klassendiagramme

Klassendiagramm Beispiel



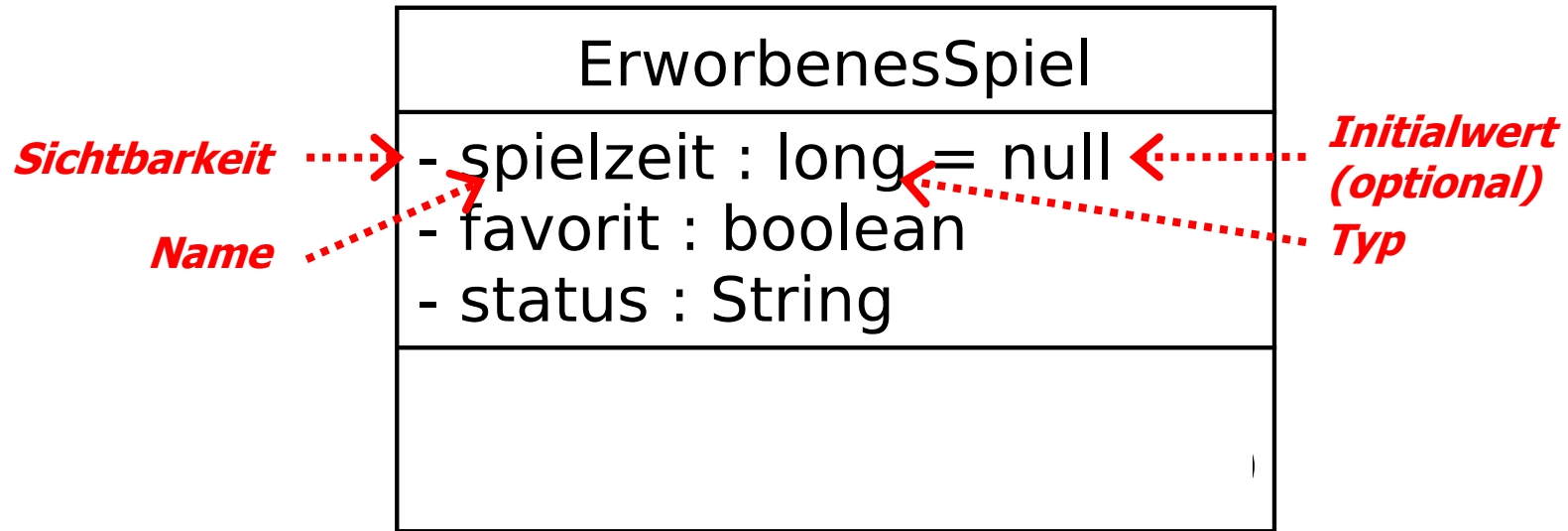
UML-Klassendiagramm

Basiselemente



UML-Klassendiagramm

Attribute



■ Notation

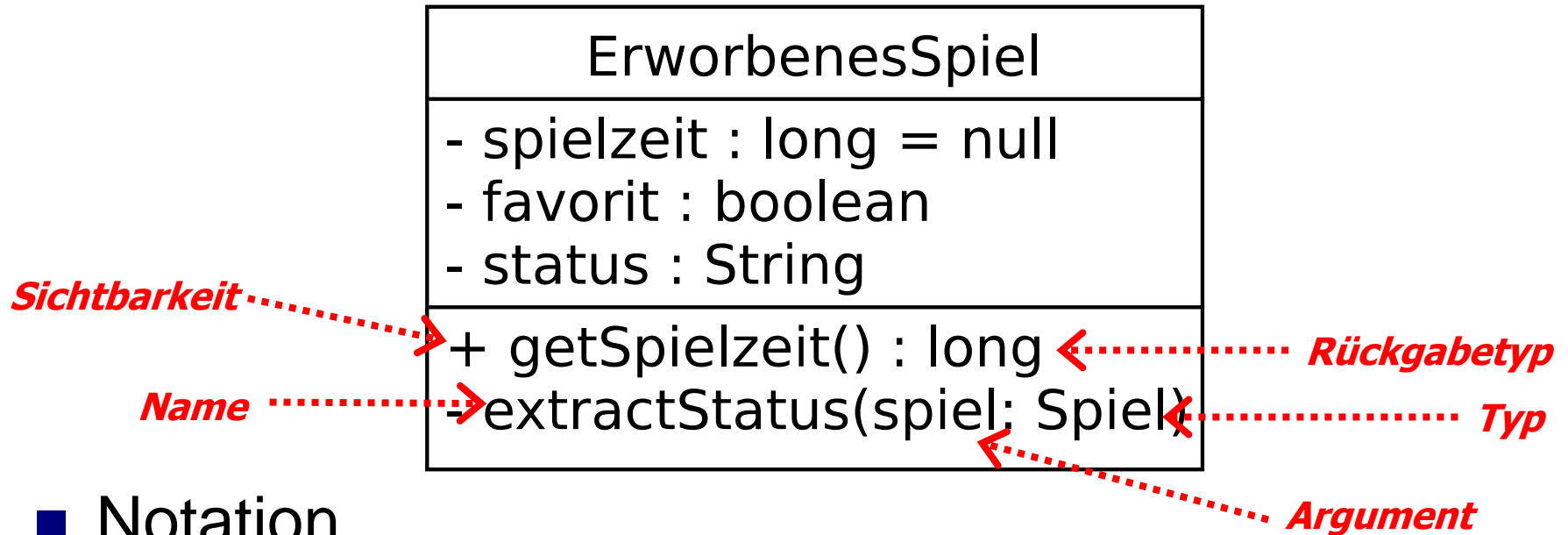
- Sichtbarkeit name: Typ = Initialwert

■ Sichtbarkeiten

- + (*Public*) - (*Private*)
- # (*Protected*) ~ (*Package*)

UML-Klassendiagramm

Operationen



■ Notation

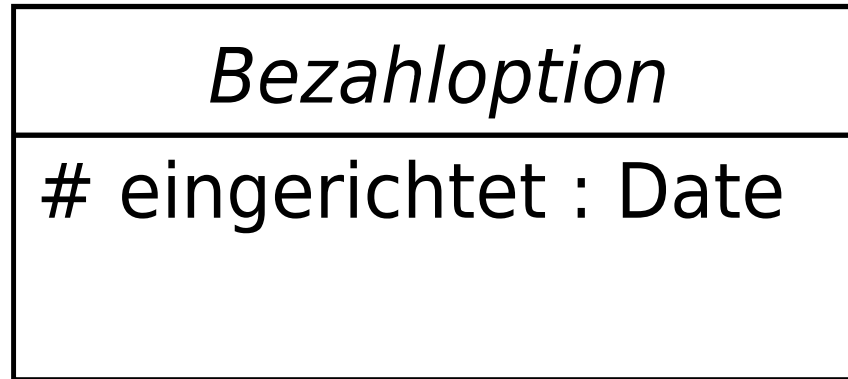
- Sichtbarkeit name (argument: Typ = Standardwert, ...): Rückgabotyp

■ Sichtbarkeiten:

- + (*Public*) - (*Private*)
- # (*Protected*) ~ (*Package*)

UML-Klassendiagramm

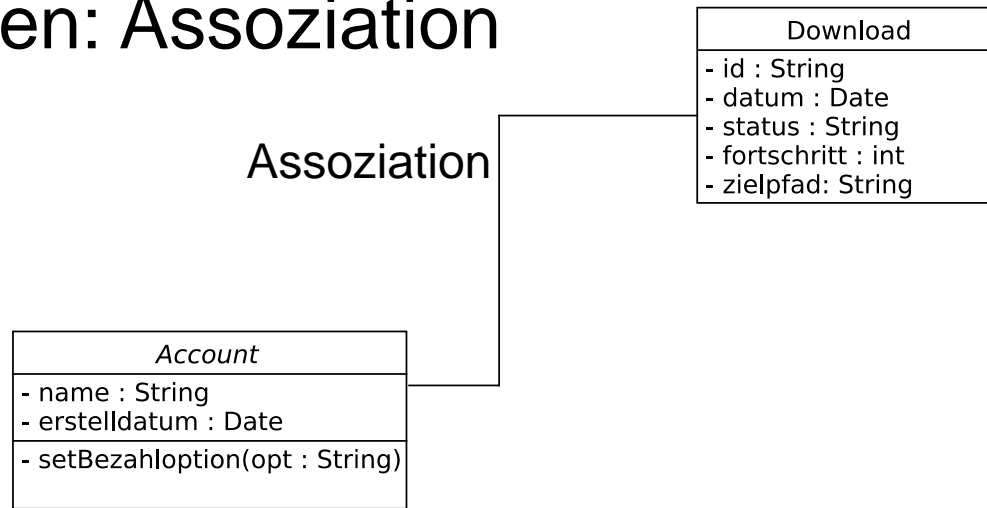
Abstrakte Klassen



- Dargestellt wie normale Klasse, aber Klassenname ist kursiv
- Abstrakt: Eine Instanziierung der Klasse ist nicht möglich

UML-Klassendiagramm

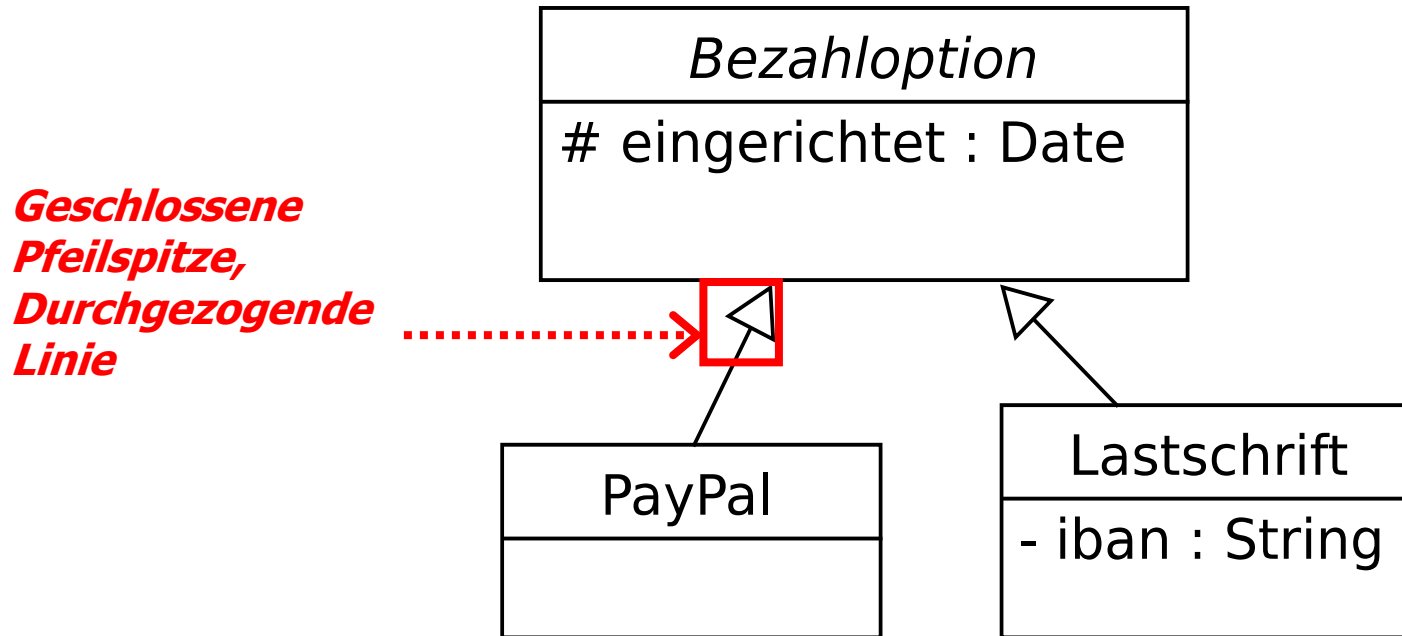
Beziehungen: Assoziation



- Zeigt Verbindung zwischen Objekten zweier Klassen
- Weitere Konzepte: Multiplizitäten, Rollenname, Sichtbarkeiten, Leserichtung

UML-Klassendiagramm

Beziehungen: Vererbung/Generalisierung

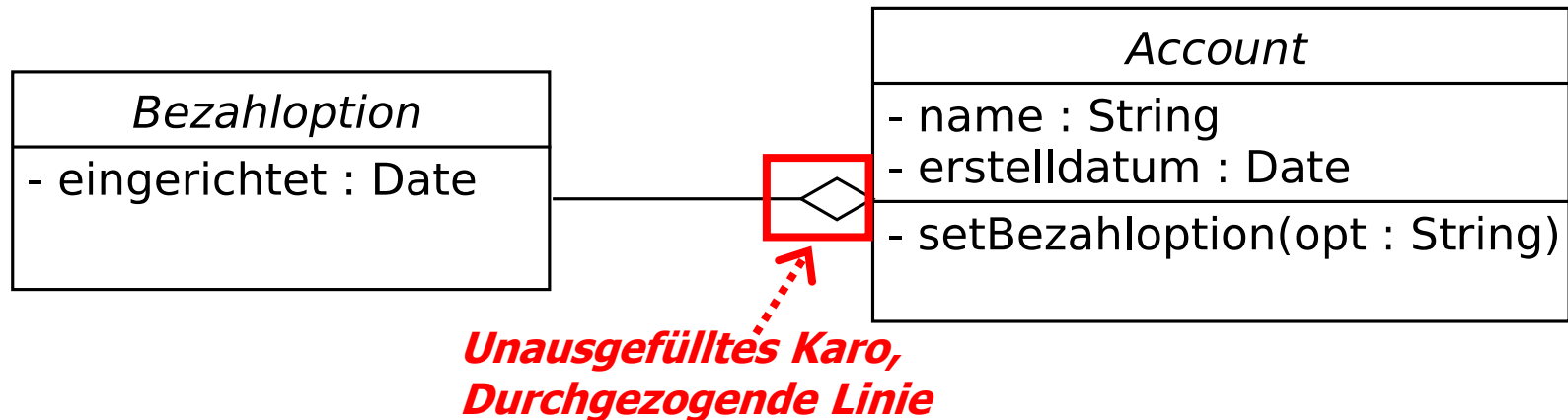


■ Ist-Ein Beziehung:

→ PayPal ist eine Bezahlung

UML-Klassendiagramm

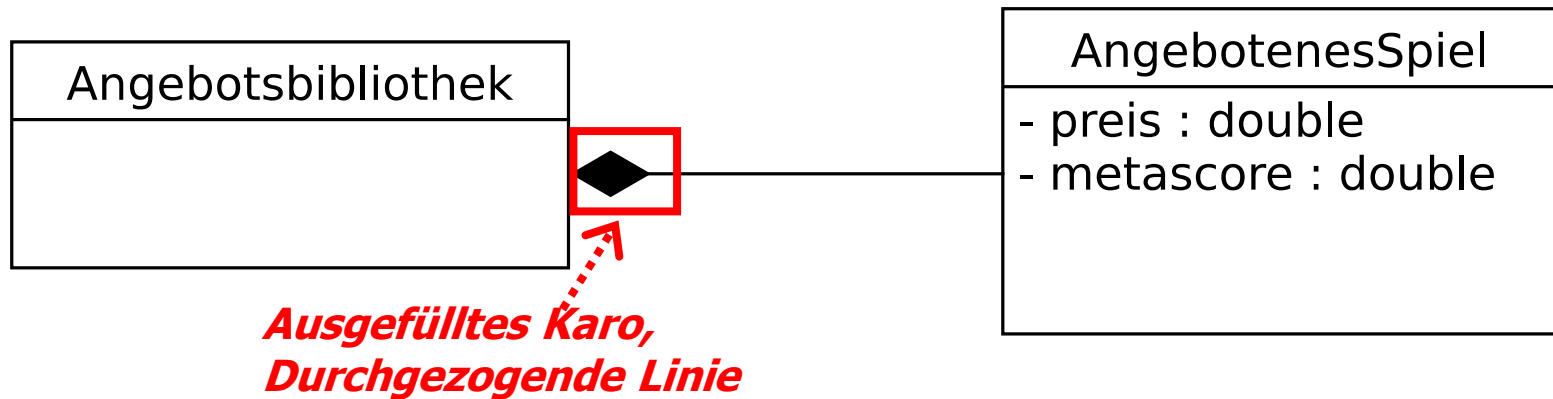
Beziehungen: Aggregation



- Unverbindlicher Kommentar
- „Ganzes-Teil-Hierarchie“
 - Account ist „Ganzes“, Bezahloption ist „Teil“
 - Bezahloption ist Teil eines Accounts

UML-Klassendiagramm

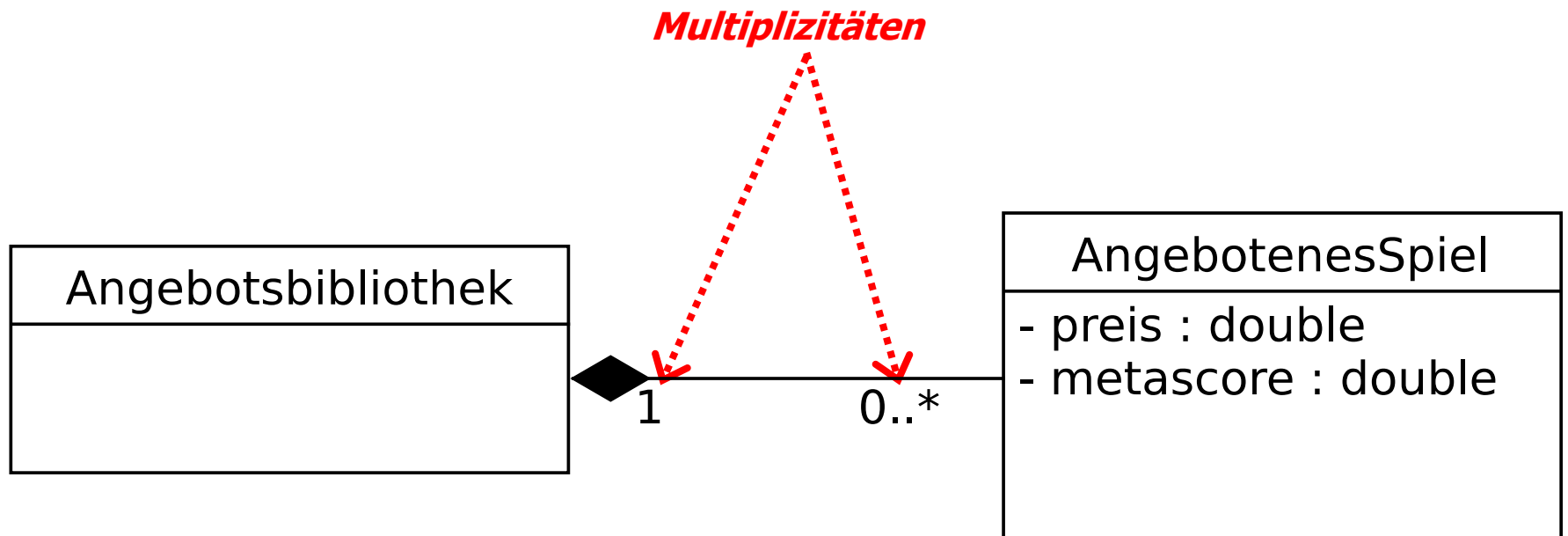
Beziehungen: Komposition



- Strenge Form der Aggregation
- „Teil“ kann nur mit „Ganzem“ existieren
- AngebotenesSpiel kann nur mit einer Angebotsbibliothek existieren
- Wenn Angebotsbibliothek gelöscht wird, dann werden auch die enthaltenen AngebotenenSpiele gelöscht

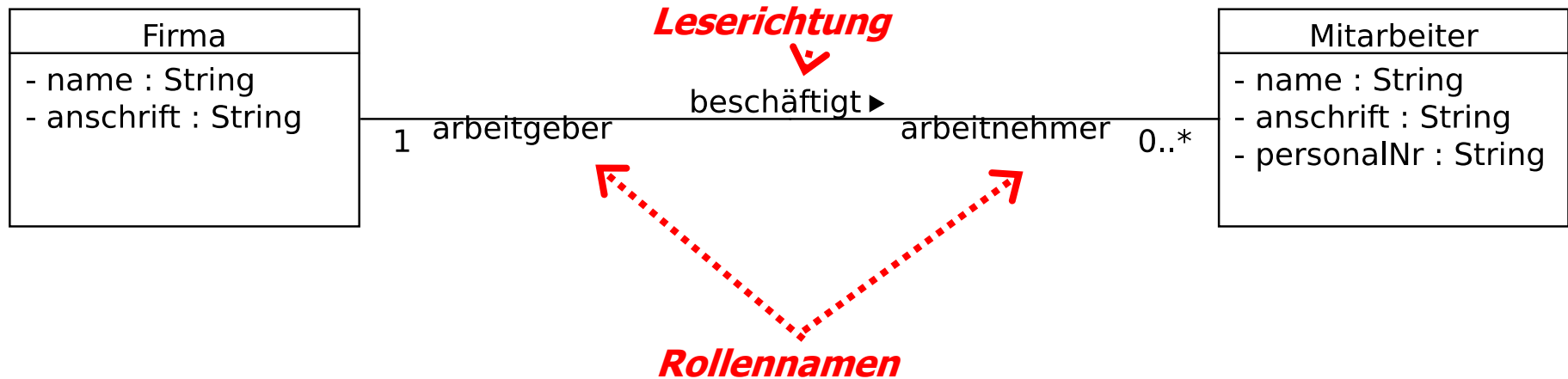
UML-Klassendiagramm

Multiplizitäten



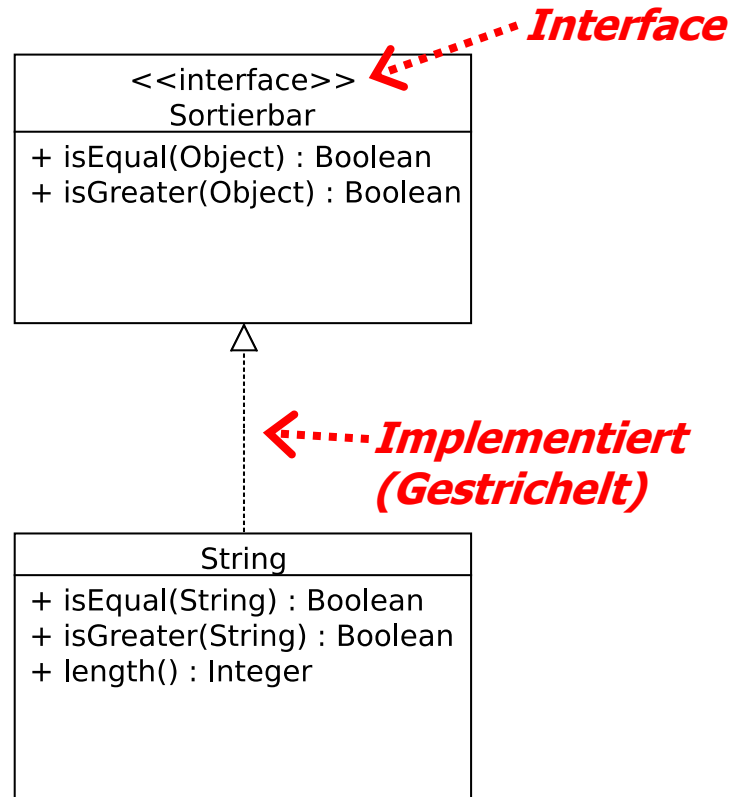
UML-Klassendiagramm

Weitere Elemente



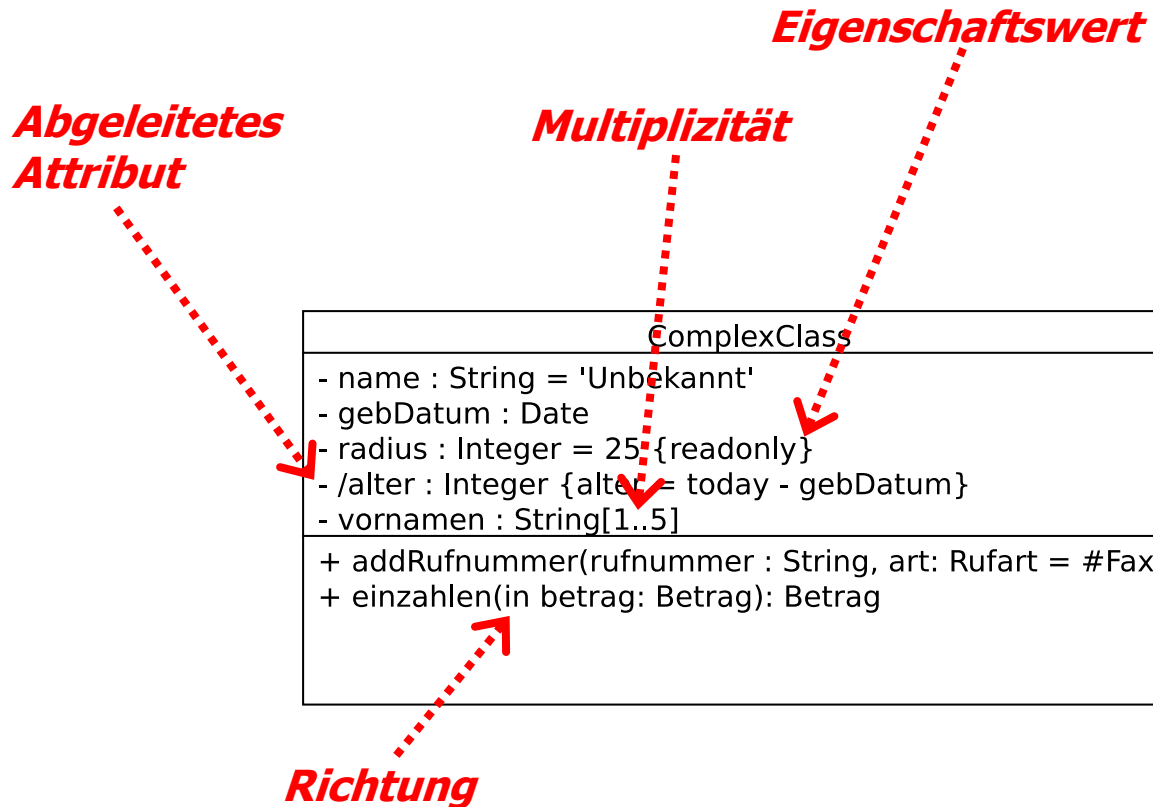
UML-Klassendiagramm

Weitere Elemente



UML-Klassendiagramm

Weitere Elemente

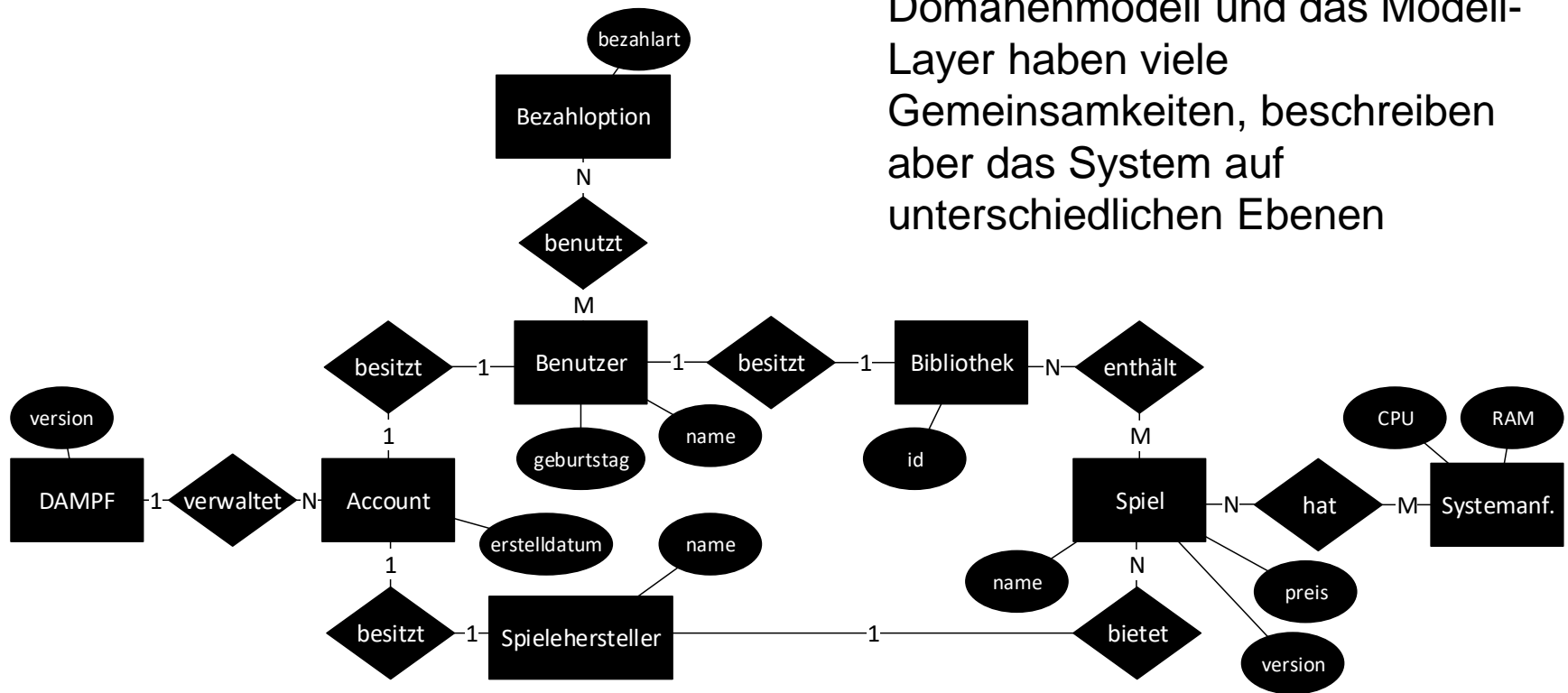




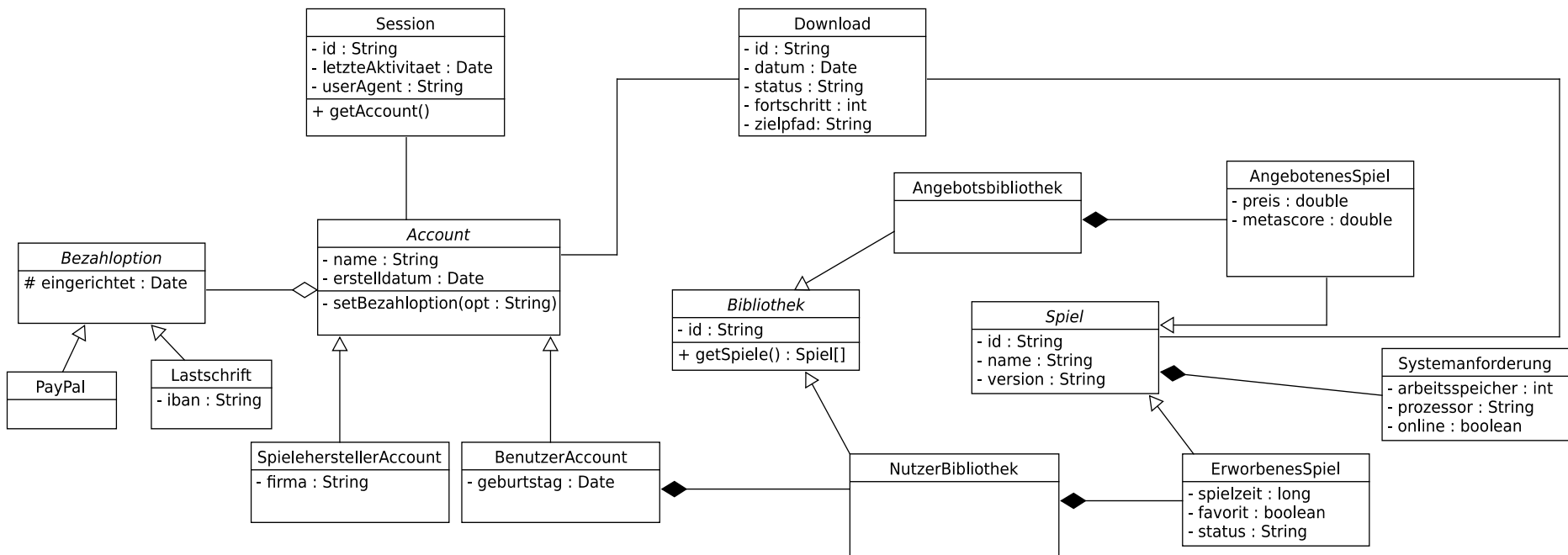
Beispiel Dampf

Domänenmodell (Analyse)

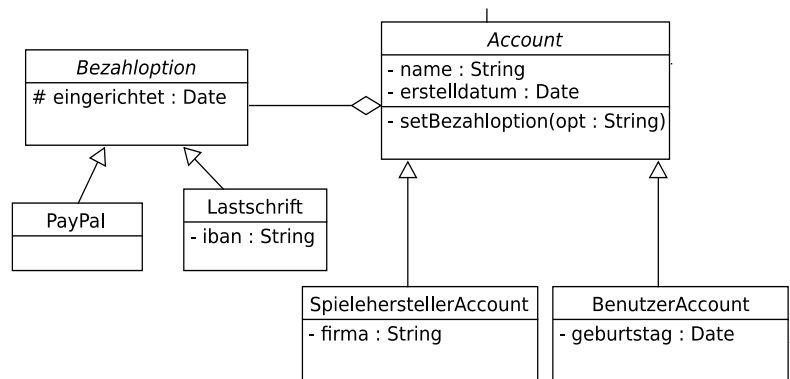
Domänenmodell und das Modell-Layer haben viele Gemeinsamkeiten, beschreiben aber das System auf unterschiedlichen Ebenen



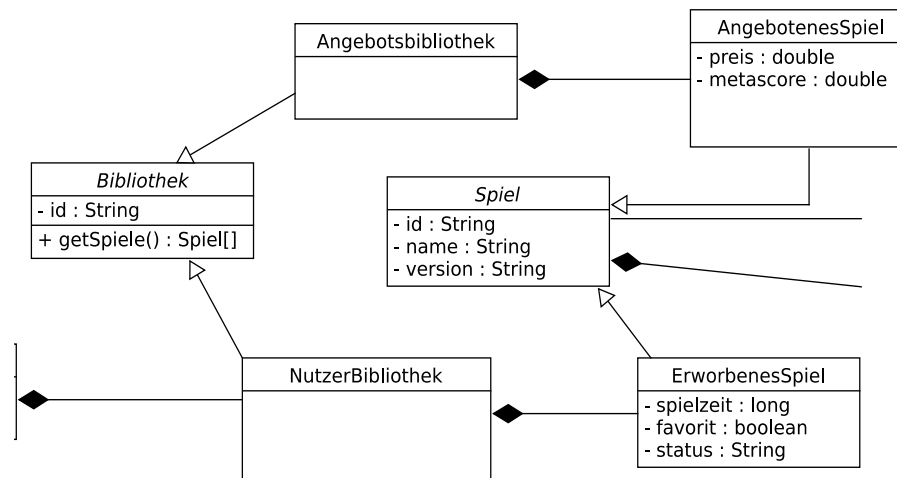
Klassendiagramm (Entwurf)



Klassendiagramm (Entwurf)



Klassendiagramm (Entwurf)





Sequenzdiagramme



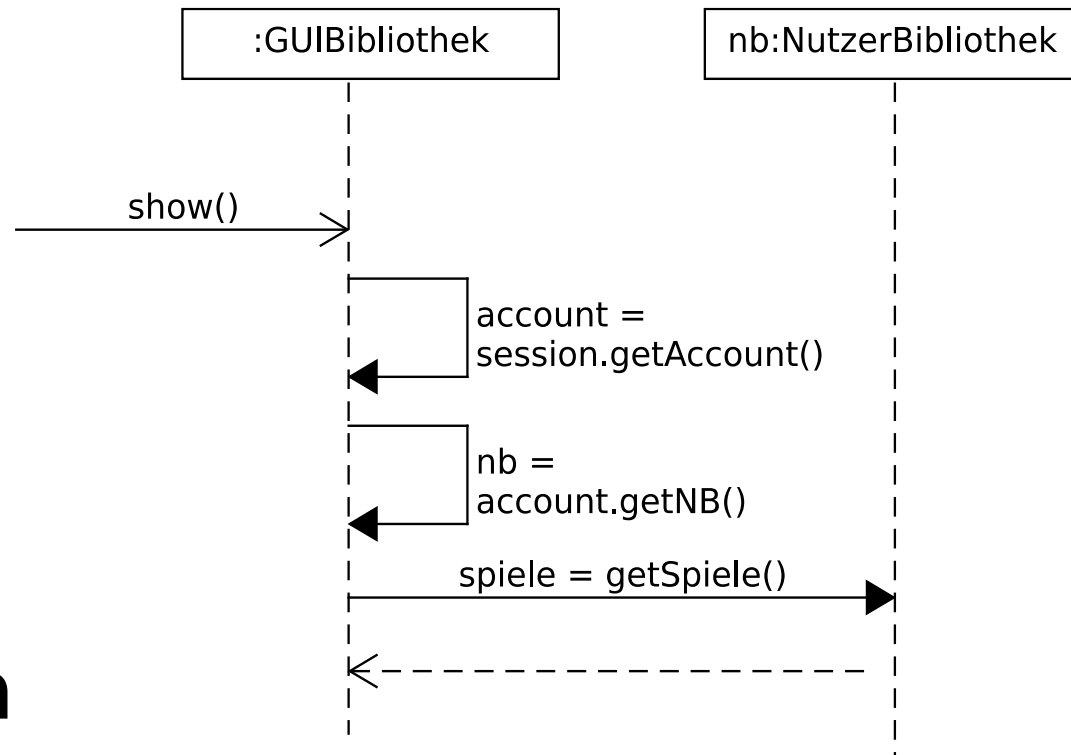
UML-Sequenzdiagramm

- Informationsaustausch zwischen Komponenten
- Fokus: zeitlicher Ablauf, Reihenfolge

UML-Sequenzdiagramm

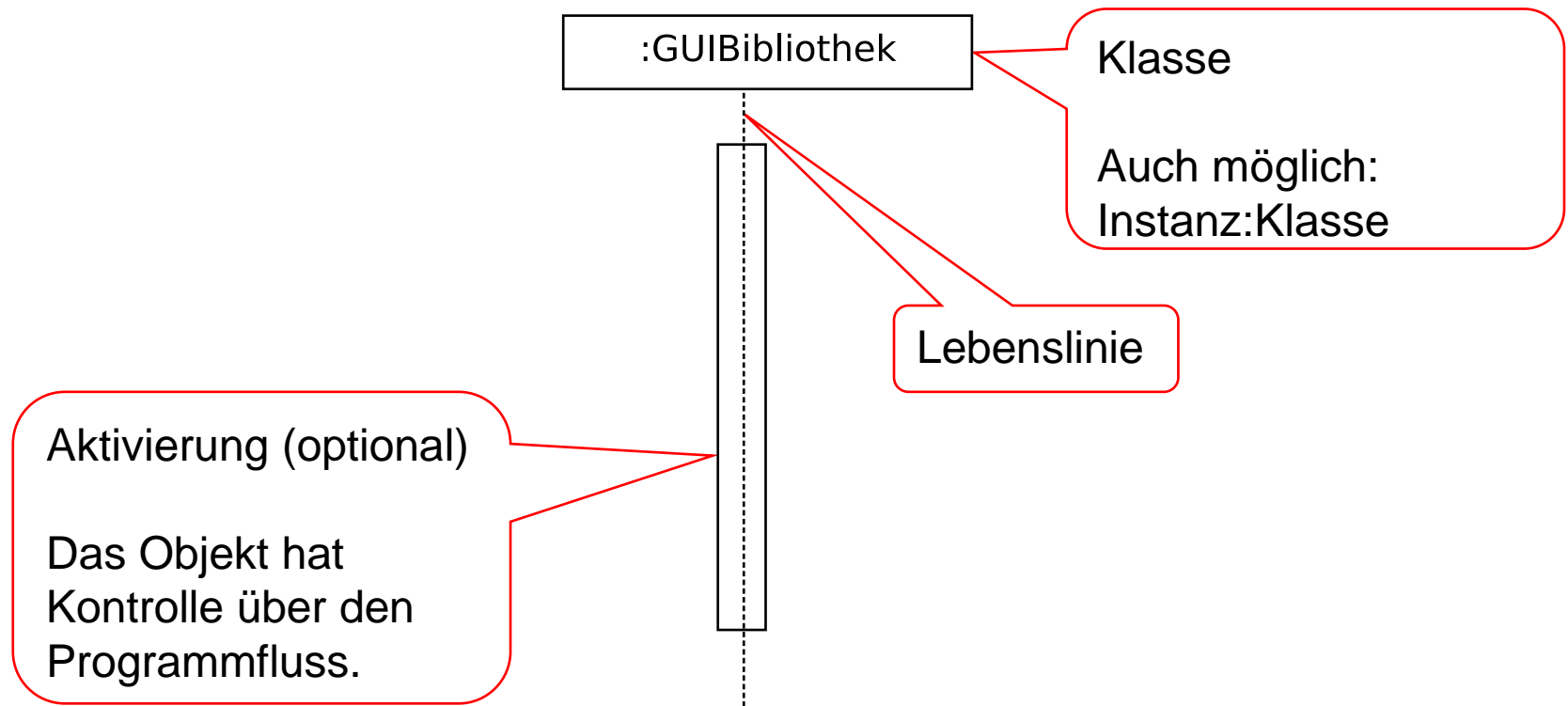
Layout des Diagrams

- **Beteiligte**
Objekte werden
waagerecht
nebeneinander
angeordnet
- **Zeit läuft von**
oben nach unten



UML-Sequenzdiagramm

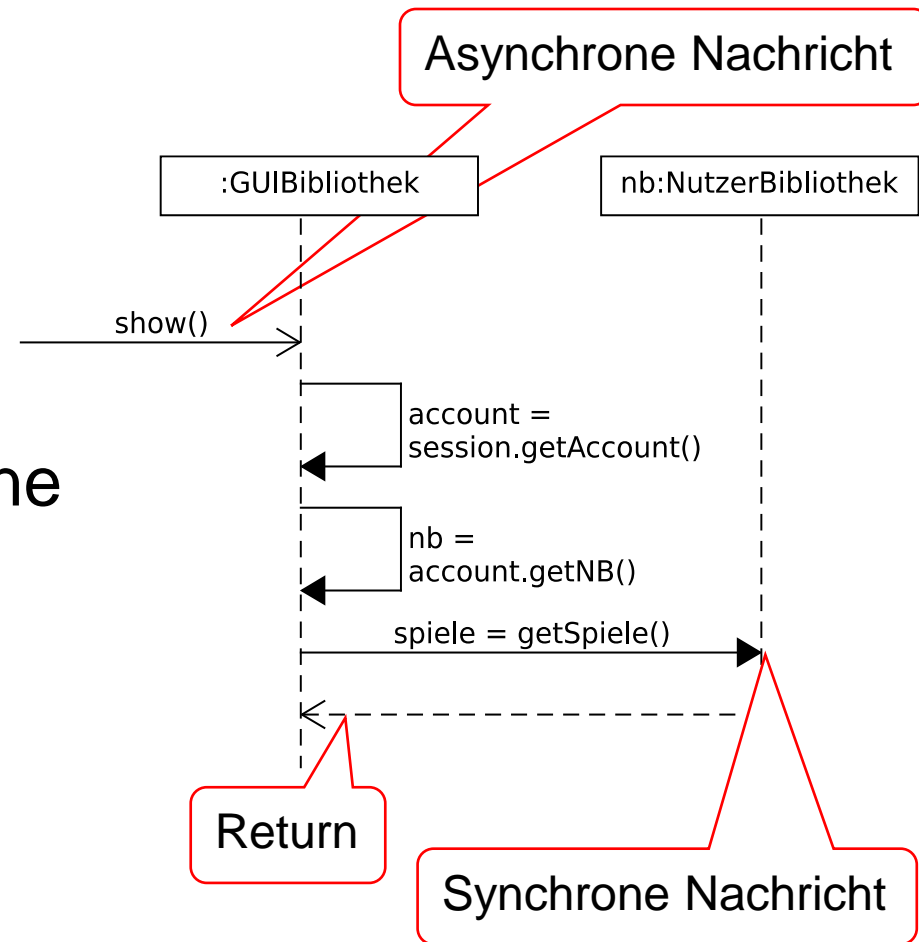
Basiselemente



UML-Sequenzdiagramm

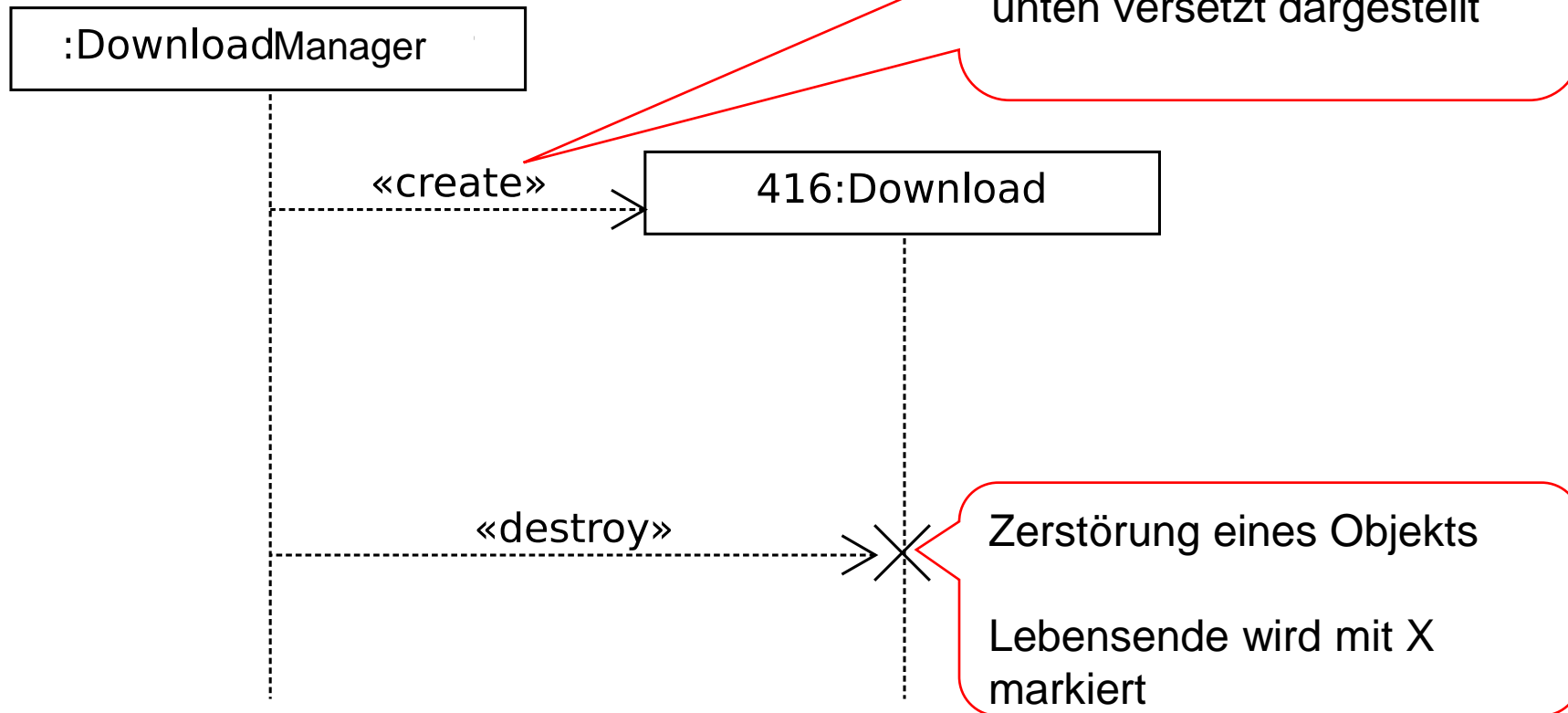
Nachrichten

- **Synchrone Nachricht**
 - Durchgezogene Linie
 - Ausgefüllte Pfeilspitze
- **Antwort / Return**
 - Gestrichelte Linie, offene Pfeilspitze
- **Asynchrone Nachricht**
 - Durchgezogene Linie
 - Offene Pfeilspitze



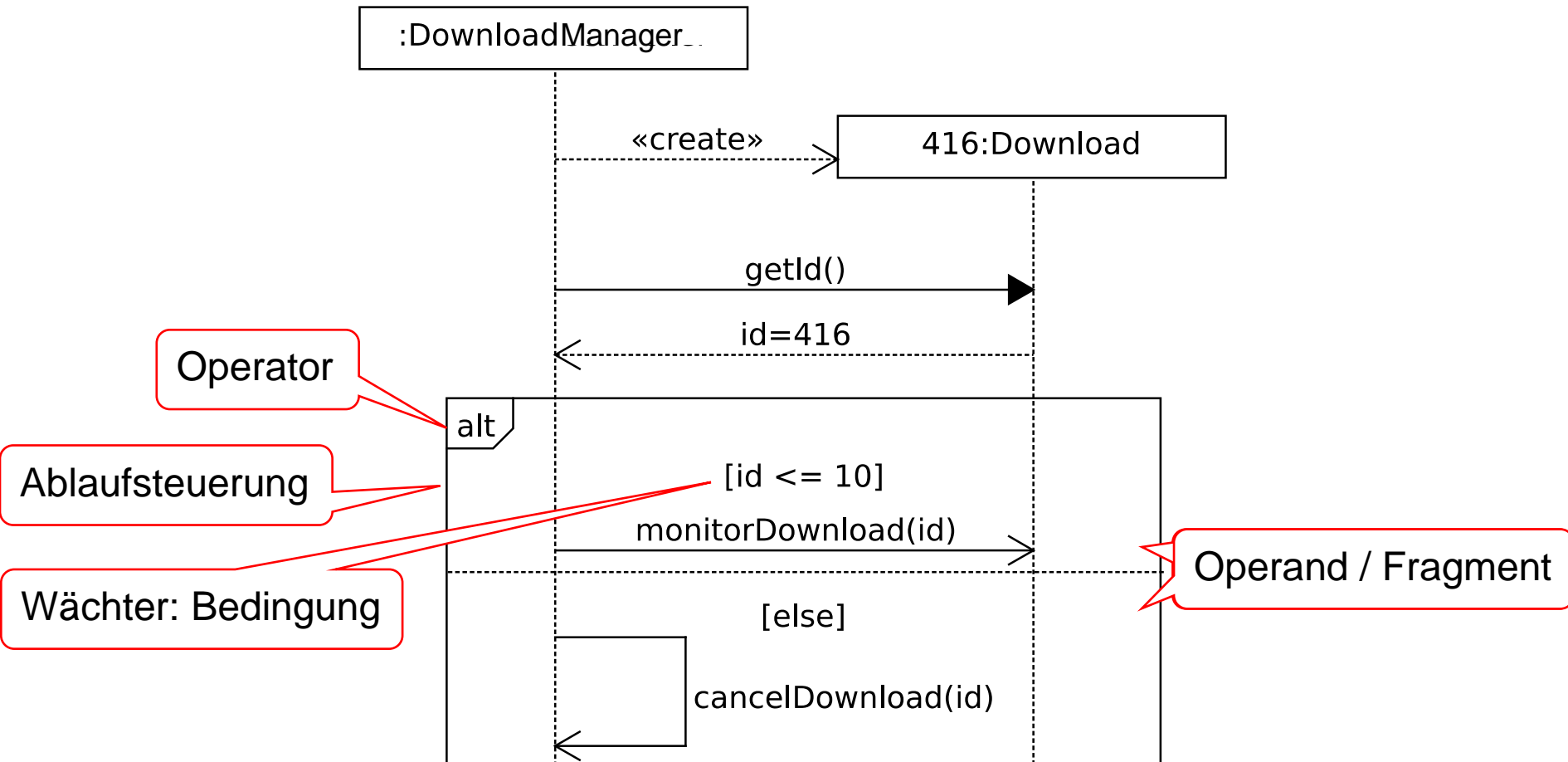
UML-Sequenzdiagramm

Objekte erzeugen und zerstören



UML-Sequenzdiagramm

Ablaufsteuerung mit Operatoren



UML-Sequenzdiagramm

Ablaufsteuerung mit Operatoren

Operator	Beding./Param.	Bedeutung
alt	[beding. 1] [beding. 2] [sonst]	Verzweigung zu einer von mehreren Möglichkeiten
loop	minint maxint [beding.]	Der Block wird als Schleife wiederholt. Schleife wird mindestens [minint], aber maximal [maxint] wiederholt falls [beding.] wahr.
break	[beding.]	Wenn dieser Block erreicht wird, endet die umgebene Schleife.
opt	[beding.]	Die Teilsequenz wird nur ausgeführt, wenn [beding.] wahr.
par		Mit diesem Operator wird angezeigt, dass die enthaltenen Teilsequenzen nebenläufig in eigenen Threads ausgeführt werden.
ref		Verweis auf weiteres Sequenzdiagramm



Verhaltensbeschreibung am Beispiel: Spiel herunterladen

