

Software Aufschrieb nur Zeug was ich nicht kann

Themen

Einführung in Java
Objektorientierte Programmierung
Modellierung, UML
Entwurfsmuster

Einführung in Java

Objektorientiert und einfach, an C++ angelehnt, beinhaltet Bibliotheken für Netzwerk und Internet, keine Zeigerarithmetik, starke Typisierung, Garbage Collector. Sicherheitsaspekte im Vordergrund (Class-Loader, Security Manager). Außerdem Architekturneutral und Portabel.

Compiler übersetzt von Sprache „X“ zu Maschinenzugabe-Prozessor +, Y, Z.

- + Optimiert für jeweiligen Prozessor, Schnell
- Programm läuft nur auf jeweiligen Prozessor, nicht zwingend identisch
- Mehrere Compiler für mehrere Prozessoren benötigt

Interpreter führt Anweisung einer Programms (nach Angabe) sofort aus

- + Nur ein Compiler für alle Prozessortypen
- + Java unabhängig von allen Plattformen exakt definiert
- Ein Java-Interpreter für jeden Prozessor-Typ nötig
- Programme laufen langsamer (Möglichkeit: Just-in-time-Compiler)

Primitive Datentypen:

Integer: char, byte, short, long, int

Gleithomme: float, double

Boolean: boolean

Probleme der Prozeduralen Programmierung:

- Jeder hat Zugriff auf globale Variablen
- Nebeneffekte von Änderungen kann zu Überblicken
- Datenkapselung besser, um keinen direkten Zugriff mehr zu erlauben

Information
Verlust

Implementierung komplexer Datenstrukturen in Modul verborgen.

kein direkter Zugriff auf Daten, nur über definierte Operationen

Implementierung der Datenstruktur bleibt unsichtbar (black box)

Klassen sind Abstrakte Datentypen

komplexe Programme in einfache Elemente zerlegen

Minimierung der Auswirkungen zu anderen Modulen

Daten und Code der Klassen gehört, sollte an einer Stelle liegen

Kohäsion ist Maß für innere Zusammenhang der Anordnungen in einer Routine

Kopplung ist Stärke der Verbindungen zwischen zwei Routinen

Starke Kohäsion, lose Kopplung

Objektorientierte Programmierung

Menschen sind mit denken in Objekten vertraut, Programmierer hätten Code schreiben, den man verstehen und nachvollziehen kann

- Attribut (Zustand): Dinge, die ein Objekt weiß
- Methoden (Verhalten): Dinge, die ein Objekt macht

Einkaufswagen
Inhalt
hinzufügen()
entfernen()
bezahlen()

Button
Beschriftung Farbe
setFarbe()
setBeschriftung()
drücken()
drückeAuflegen()

Klasse beschreibt Menge von Objekten mit gemeinsamen Eigenschaften und ähnlichem Verhalten

Enthält eine vollständige Beschreibung von:

- Den einzelnen Datenelementen (Attributen) eines Objekts
- Den Operationen, die das Objekt ausführen kann
- Die Art, wie die Attribute und Methoden angesprochen werden können

Jedes Objekt kann eigene Daten haben, erfüllt aber immer die gleichen Operationen

Sichtbarkeit: ohne Angabe grundsätzlich in ganzen Paket sichtbar

- public: sichtbar für alle andere Klasse
- private: sichtbar nur innerhalb der Klasse
- protected: sichtbar innerhalb der Klasse und in allen Unterklassen

Je nach Sichtbarkeit Verwendung von getter() und setter() Methoden

Konstruktor werde beim Erzeugen eines Objekts einer Klasse über das Schlüsselwort new aufgerufen.

Klassenattribute mit static gekennzeichnet, sind für alle Objekte einer Klasse gleich

Vererbung um aufbauend auf existierende, neue Klassen zu schaffen

Beziehung ist dauerhaft, neue Klasse kann Erweiterung oder Einschränkung sein

Sichtbarkeiten der Oberklasse bleiben der Unterklasse erhalten

Ersatzmechanismus: Schnittstellen (Interfaces)

Klassen, die nicht instanziiert werden sollen als abstract definieren, können dann nur noch als Referenz verwendet werden

Abstrakte Klassen können abstrakte Methoden haben: keinen Körper, deklariert nur Zugriffsmodifizierer, Rückgabetype, Methodensignatur
konkrete Unterklassen müssen abstrakte Methoden implementieren.

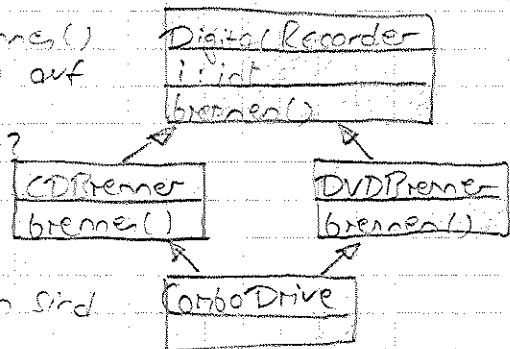
Polymorphismus: Objekt und Referenz nicht gleich: Animal a = new Dog();

Jede Klasse, die nicht explizit von einer anderen Klasse erbt, erbt von der Klasse Object; Methoden wie equals(), getClass(), toString(), ...

Software Aufschrieb nur Zeug was ich nicht kann

Problem bei Mehrfachvererbung: Welche `brennen()` Methode wird ausgeführt, wenn man `brennen()` auf dem `ComboDrive` ausführt? Was passiert, wenn `ComboDrive` beide Werte von `i` braucht?

- keine Mehrfachvererbung in Java möglich
- Alternative: Schnittstelle (Interface)



Schnittstelle gibt an, welche Methoden vorhanden sind oder vorhanden sein müssen

- Werden mit Schlüsselwort `implements` implementiert
- Klasse kann mehrere Schnittstellen implementieren
- Methoden in Java-Schnittstellen `public` und `abstract`

Folien 720-736 noch nacharbeiten wenn genug Zeit

Exceptions: Programme müssen auf Fehler reagieren

`throw`, `try`- und `catch`-Blöcke, finally optional

Alle Fehlerklassen in Java von `Throwable` abgeleitet

- **Error**: Schwerster Fehler im System, normalerweise Programmstopp
- **Exception**: Wichtig für Programmierer: weitere Aufteilung:
 - **Runtime Exception**: Laufzeitfehler während Programmverlauf: Fehlerhafte Typkonvertierung, Zugriff über `array`-grenze hinaus, ...
 - **IO Exception**: Fehler durch ungünstige Umstände: ungenügend Systemressourcen, Fehler beim Lesen einer Datei, ...

UML Klassendiagramme

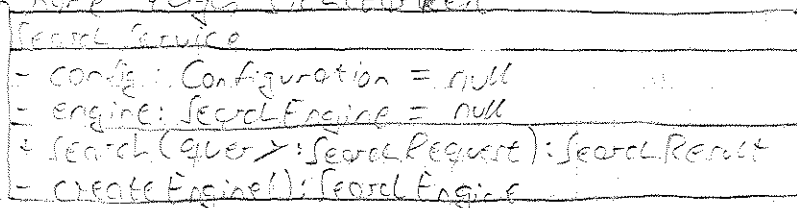
Unified Modeling Language ist grafische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Software-Teilen und anderen Systemen

Ein Klassendiagramm ist ein Strukturdiagramm zur grafischen Darstellung (Modellierung) von Klassen, Schnittstellen sowie deren Beziehungen

Notation

- Klassenname ist obligatorisch
 - Attribute entsprechen Klassenvariablen, Typ nach Doppelpunkt
 - Klassenoperationen (Methoden), Typ der Rückgabewerts nach Doppelpunkt
- Symbole vor Attribut- oder Operationen zeigen Sichtbarkeit

- "+" für public
- "-" für private
- "#" für protected
- "~" für package



Signatur einer Operation besteht aus

- Name
- Parameterliste (optional)
- Rückgabewert (optional, kann void in UML)

Behandlung der Parameter einer Operation

- in: Übergebener Parameter wird nur gelesen (standard)
- inout: Übergebener Parameter wird gelesen, verarbeitet und beschrieben
- out: Übergebener Parameter wird beschrieben, ohne ihn vorher zu lesen
- return: Übergebener Parameter wird als Rückgabewert zurückgegeben

Attribute, Parameter und Operationen können in geschweiften Klammern mit zusätzlichen Eigenschaften versehen werden

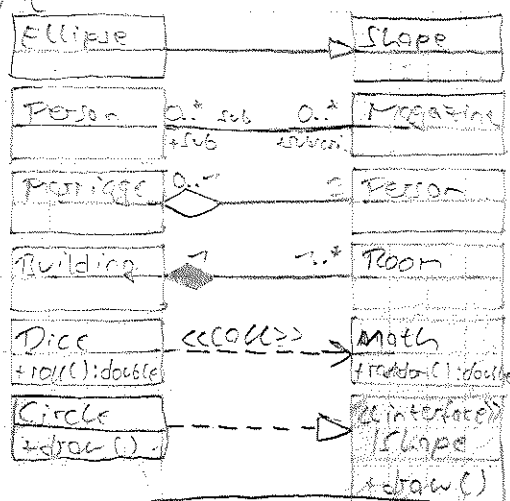
- ordered: Werte sind sortiert
- unique: Werte enthalten keine Duplikate (standard)
- readonly: Wert ist nur lesbar
- redefines: Überschreibt geerbte Operationen
- query: Operation ändert nicht den Zustand des Systems

Statische Elemente werden unterstrichen dargestellt

Abstrakte Elemente werden kursiv dargestellt

Beziehungen zwischen Klassen in UML

- Generalisierung (Inheritance)
Ist-ein-Beziehung
- Assoziation (Association)
Ist-ein-Beziehung
- Aggregation (Aggregation)
Ist-Teil-von-Beziehung
- Komposition (Composition)
Sonderfall Aggregation
- Abhängigkeit (Dependency)
Eine Klasse benutzt die andere
- Realisierung (Implementation)
Interface zu implementierender Klasse



Beispiel für UML Klassendiagramm auf Folien 758-762

Software Aufschrieb nur Zeug was ich nicht kann

Entwurfsmuster

OO-Prinzipien

- Trenne Aspekte die sich ändern von jenen die konstant bleiben
- Entwürfe offen für Erweiterung, geschlossen für Veränderungen
- Programmiere auf Schnittstellen, nie auf Implementierung
- Eine Klasse, eine Verantwortlichkeit
- Komposition (hat-ein) ist der Vererbung (ist-ein) vorzuziehen

Gewisse Lösungen für häufig wiederkehrende Probleme, Entwurfserfahrungen inkapsulieren, Ein gutes Entwurfsmuster sollte:

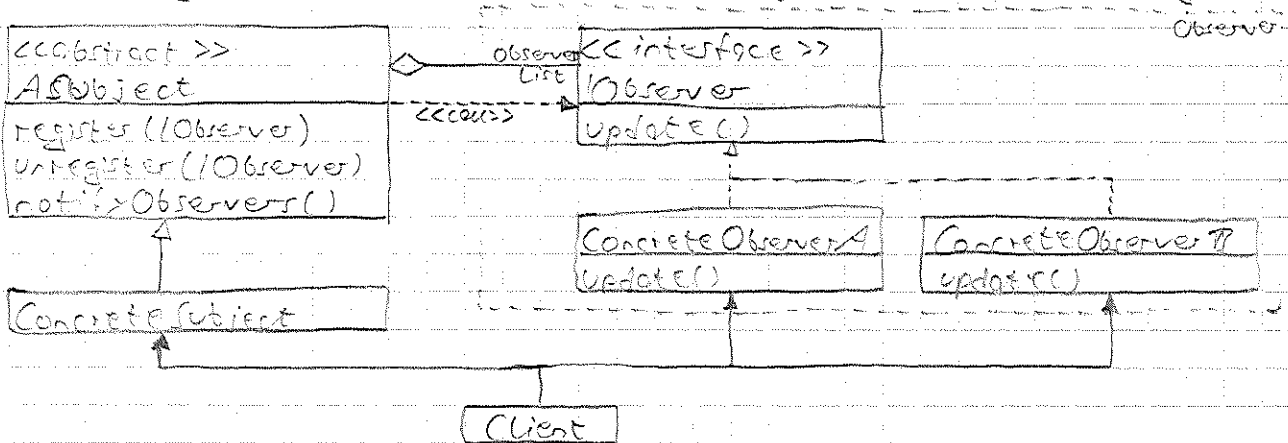
- ein oder mehrere Probleme lösen
- ein erprobtes Konzept bieten
- über das rein Offensichtliche hinausgehen

Observer-Pattern (Diagramm Folie 727)

Änderung des Zustands eines Objekts führt dazu, dass alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden

- + Daten im Gesamtsystem bleiben ~~konsistent~~ konsistent
- + Flexibel, Modular: mehrere Observer oder mehrere Subjects
- + Minimaler Änderungsaufwand
- Aktualisierungskosten bei großen Systemen (oder Rekursion!)
- Es kann vergessen werden einen Observer beim Subject abzumelden (Mehrfachanmeldung $\rightarrow$ Mehrfachbenachrichtigung)

Anwendung in GUIs, Datensätzen mit mannigfaltigen Visualisierungen

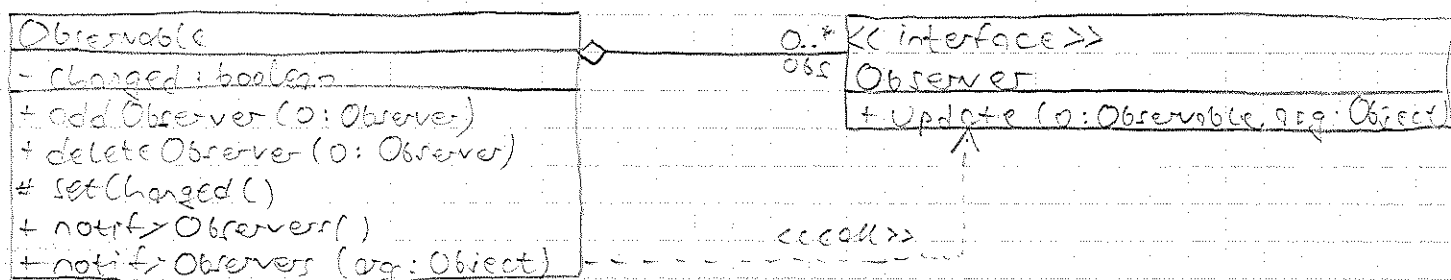


Rein Push-Modell: Übergibt das Subject der `update()` Methode detaillierte Informationen über die Änderung als Parameter

Rein Pull-Modell: erhält der Observer nur eine minimale Benachrichtigung und muss sich die benötigten Informationen selber aus dem Subject holen

Verwaltungscode im konkreten Subject, Abstrakter Subject-Superklasse oder in Extra-Helferklasse

In Java ist `java.util.Observable` und `java.util.Observer` schon eingebaut



Singleton-Muster : Sicherstellen, dass eine Klasse genau ein Exemplar besitzt. Er wird der Konstruktor privatisiert, kann also nur noch in der Singleton-Klasse selbst aufgerufen werden.
Variante:

- Eager Loading : einfach, thread sicher, aber verfrühte oder sogar unnötige Instanziierung
- Lazy Loading : Instanz wird erst erstellt wenn gebraucht, nicht Thread sicher
- Thread-Sicherheit : Synchronisierte getInstance-Methode (locken) synchronisierter Block, um new Singleton() mit doppeltem null-check

Strategie-Muster : Definiere eine Familie von Algorithmen, kapsle jeden einzelnen und mache sie austauschbar.

Das Strategiemuster ermöglicht es, den Algorithmus unabhängig von ihm nutzenden Clients zu variieren.

Dekorier-Muster : Erweitere ein Objekt dynamisch um Zuständigkeiten. Dekorierer bietet eine flexible Alternative zur Unterklassenbildung, um die Funktion einer Klasse zu erweitern.

Fabrik-Methode : Definiere eine Klassenmethode mit Operationen zum Erzeugen eines Objekts, aber lasse unterklasse entscheiden, von welcher Klasse das zu erzeugende Objekt ist.

- Wiederverwendbarkeit : Verarbeitungsschritt für neue Programme kann wieder verwendet werden
- Konsistenz : Verarbeitungsschritt kann in gleicher Stelle gewartet und erweitert werden
- Erweiterbarkeit : Neue Office-Funkts können hinzugefügt werden, ohne bestehende Code zu verändern

Abstrakte Fabrik-Methode auch möglich : Bietet eine Schnittstelle zum Erzeugen von Familien verwandter oder voneinander abhängiger Objekte, ohne ihre konkrete Klasse zu benennen.

Gemeinsamkeiten : Beide Muster entkoppeln den Client von konkreten Typen. Client kennt nur die Schnittstelle zur Fabrik und der Produkten.

Abstract Factory

- Ganze Produktfamilie wird erstellt
- Breite Schnittstelle
- Schnittstelle (Abstract Factory) ist oft nur ein Interface
- Benutzt Factory Methoden aber ohne generischen Code

Factory

- Ein-Produkttyp wird erstellt
- Schmale Schnittstelle
- Schnittstelle ist oft abstrakt und enthält generischen Code

Kompositum-Muster : Füge Objekte zu Baumstrukturen zusammen, um Teil-Ganzes-Beziehungen zu repräsentieren. Clients wird ermöglicht, einzelne Objekte sowie Kompositionen von Objekten einheitlich zu behandeln.

Kommando-Muster : Kapsle einen Befehl als ein Objekt. Dies ermöglicht es, Clients mit verschiedenen Anfragen zu parametrisieren, Operationen in eine Schlange zu stellen, ein Logbuch zu führen und Operationen rückgängig zu machen.