INTERFACEBASIERTE PROGRAMMIERUNG

Studientag 01853

Moderne Programmiertechniken und -methoden

FRAGILE BASE CLASS PROBLEM

- bei Vererbung bestehen Abhängigkeiten zwischen abgeleiteter Klasse und der Basisklasse

- Änderungen der Basisklasse können unvorhergesehene Verhaltensänderungen hervorrufen

INTERFACES

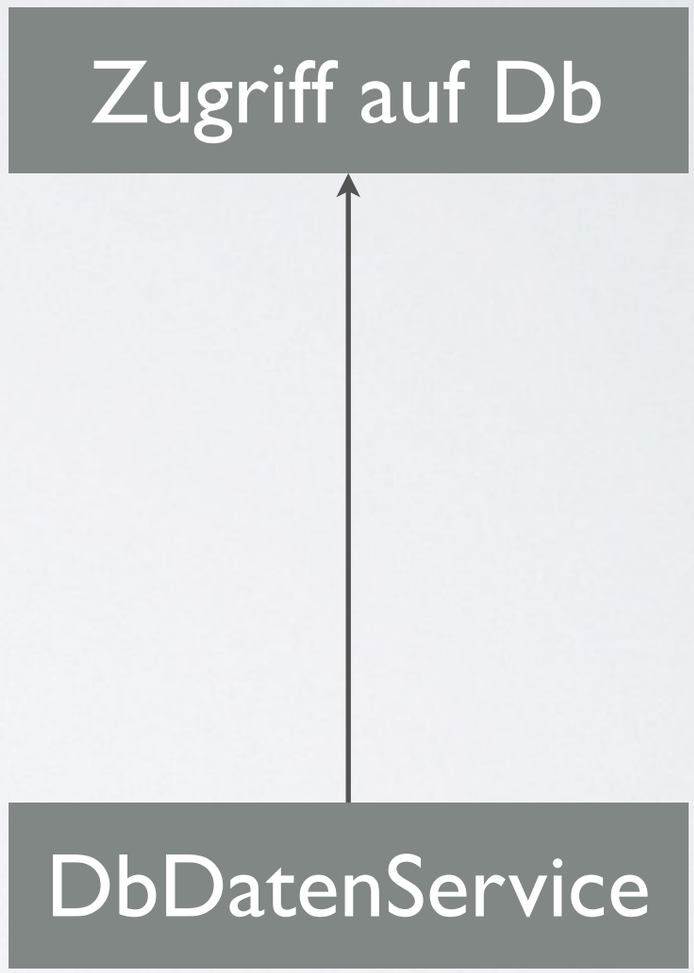
- IEEE: „eine gemeinsame (Modul-)Grenze, über die hinweg Information gereicht wird“

- umfassen Deklarationen von Methoden, Variablen, Konstanten und Typen (nicht alle in jeder Sprache verfügbar)

- also: was aber nicht wie

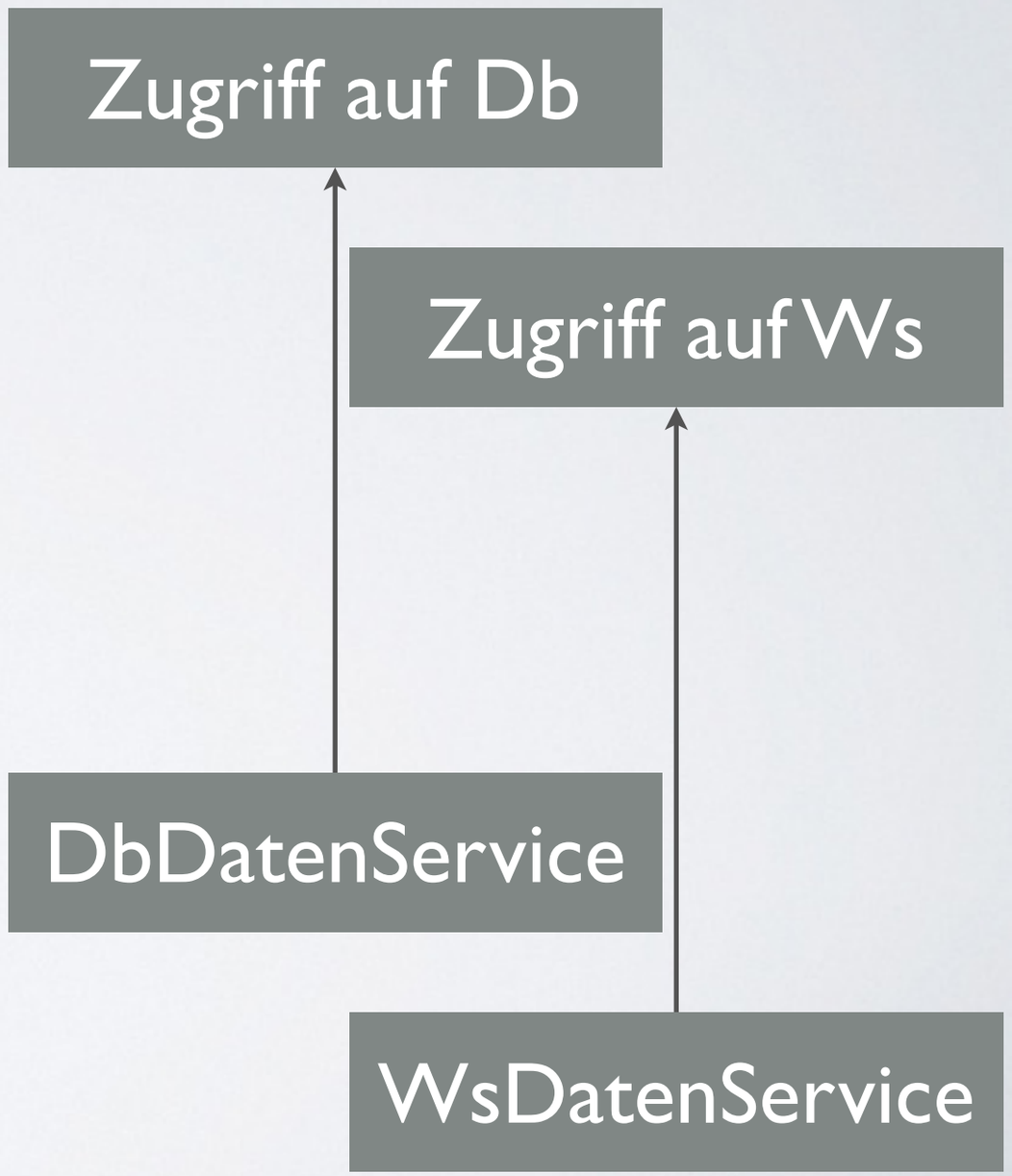
BEISPIEL

- ein Programm lädt Daten aus einer Datenbank



BEISPIEL II

- ein Programm lädt Daten aus einer Datenbank
- als neue Funktion sollen nun Daten auch über einen Webservice geladen werden

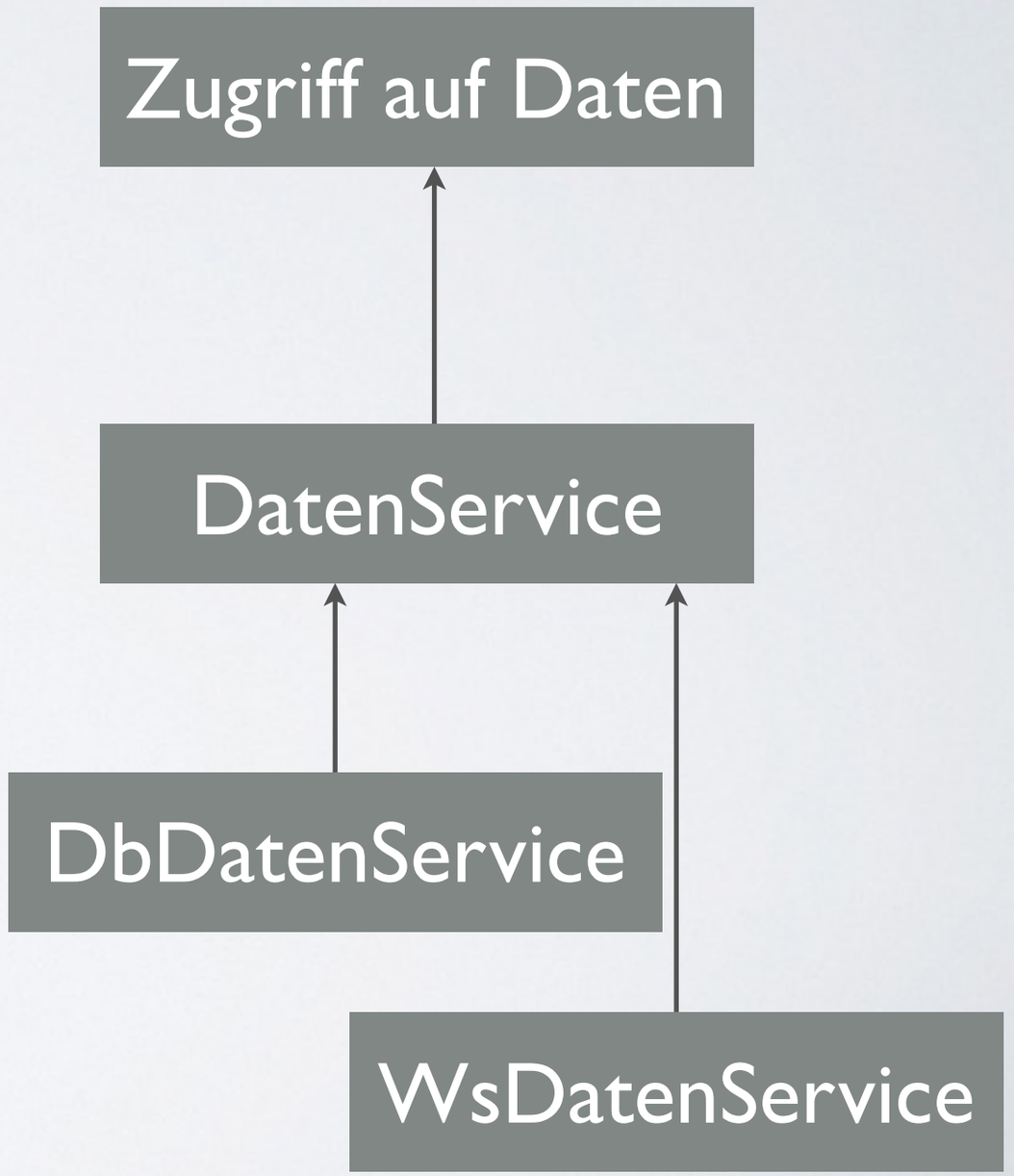


BEISPIEL III

- ein Programm lädt Daten aus einer Datenbank

- als neue Funktion sollen nun Daten auch über einen Webservice geladen werden

- mit Interfaces wir das Laden von Daten abstrahiert



INTERFACES (CONT.)

- Information hiding
  - Entwurfsentscheidungen werden „hinter Interfaces verborgen“
  - Vermeidung von Seiteneffekten
- nicht an OOP gekoppelt (s. Microsoft COM)

TYPKONFORMITÄT

- strukturelle
  - syntaktische Gleichheit der Deklarationen
  - flexible und automatische Modularisierung
  - aber: kann sich zufällig ergeben (bei ungleicher Semantik)
- nominale
  - Implementierung von Interfaces nur mit expliziter Angabe
  - erzwingt strukturelle Typkonformität

JAVA

- erstes verbreitetes Aufkommen von Interfaces als Typen
- unterschiedliche Verwendung
  - Marker-Interface (Serializable)
  - MVC / Swing (Listener)
  - Collection API

C#

- Besonderheit im Vergleich zu Java: explizite Interfaceimplementierung
  - eine Klasse implementiert eine Methode, deren Signatur in mehreren implementierten Interfaces identisch ist
  - für jedes Interface kann eine eigene Implementierung der Methode angegeben werden, die dann nicht mehr Teil des Klasseninterfaces ist
- aufrufende Variable muss mit Interface typisiert sein um auf eine der Methoden zuzugreifen

C++

- kein Interface-als-Typ Konzept
- kann aber durch abstrakte Klasse und Mehrfachvererbung umgesetzt werden
  - semantischer Aspekt (Generalisierung oder Interface) geht aber verloren
- Vorteil: direkte Standard-Implementierung von neuen Interfacemethoden

EIGENSCHAFTEN

- klassisch: aufrufende und aufgerufene Seite
  - komponentenbasiert: angebotene und benötigte Interfaces
- totale und partielle Interfaces
  - Interface Segregation: Beschränkung der Abhängigkeiten auf ein Minimum
- öffentliche und veröffentlichte Interfaces

WIE VIELE INTERFACES?

```
import com.example.Person;

public class Student implements Person, Serializable {
  public String getName() {
    return name;
  }
  protected String formatName() {
    ...;
  }
  private String name;
}
```

WIE VIELE INTERFACES?

```
import com.example.Person;

public class Student implements Person, Serializable {
  public String getName() {
    return name;
  }
  protected String formatName() {
    ...;
  }
  private String name;
}
```

WIE VIELE INTERFACES?

```
import com.example.Person;

public class Student implements Person, Serializable {
  public String getName() {
    return name;
  }
  protected String formatName() {
    ...;
  }
  private String name;
}
```

WIE VIELE INTERFACES?

```
import com.example.Person;

public class Student implements Person, Serializable {
  public String getName() {
    return name;
  }
  protected String formatName() {
    ...;
  }
  private String name;
}
```

WIE VIELE INTERFACES?

```
import com.example.Person;

public class Student implements Person, Serializable {
  public String getName() {
    return name;
  }
  protected String formatName() {
    ...;
  }
  private String name;
}
```

ANZEICHEN

- Klassen implementieren Interfaces

- Variablen werden mit Interfaces als Typ deklariert

  ➡ lose Kopplung

KLASSIFIZIERUNG

- Allgemeinheit
  - allgemeine Interfaces (total)
  - kontextspezifische Interfaces (partiell)
- Nutzen
  - anbietende Interfaces (Services)
  - ermöglichende Interfaces („Don‘t call us, we‘ll call you“)

KLASSIFIZIERUNG II

	allgemein	kontextspezifisch
anbietend	idiosynkratisches I. Familieninterface	Client/Server
ermöglichend		Server/Client Server/Item

IDIOSYNKRATISCHE INTERFACES

- allgemein, anbietend
- wird nur von einer Klasse implementiert
  - in aktuellen OO-Sprachen i.d.R. durch Klasseninterface ersetzt

FAMILIENINTERFACES

- allgemein, anbietend
- Differenzierung unterschiedlicher Implementationen
- Bsp.: Repositories zum Speichern von Objekten:
  - in einer Datenbank
  - als Textdatei
  - als XML-Datei

CLIENT/SERVER INTERFACES

- kontextspezifisch, anbietend
- aufrufende und profitierende Klasse: Client
- aufgerufene und leistende Klasse: Server
- nur partielles Interface
- vgl: Familieninterface als totales Interface

SERVER/CLIENT

- kontextspezifisch, ermöglichend
- aufgerufene und leistende Klasse: Client
- aufrufende Klasse: Server
- Beispiele: Listener, java.lang.Runnable

SERVER/ITEM

- kontextspezifisch, ermöglichend
- initiierende Klasse: Client
- aufrufende Klasse: Server
- aufgerufene Klasse: Item
- Beispiel: java.lang.Comparable, java.awt.print.Printable, Marker-Interfaces

DEPENDENCY INJECTION (DI)

- Problem in der Interfacebasierten Programmierung: Objekterzeugung schafft Abhängigkeit zu konkreten Klassen
- Lösung: Auslagern der Erzeugung und übergeben (injizieren) in benutzendes Objekt über
  - Konstruktor
  - Setter-Methode
  - Interface-Methode

DEPENDENCY INJECTION II

- Zusammenführen der Objekte z.B. durch Assembler
  - programmiert oder konfigurierbar
- nicht anwendbar bei
  - Objekterzeugung nach Laufzeitbedingungen
  - unklarer Zeitpunkt der Zuweisung
  - Zuweisung temporärer Variablen

DEPENDENCY INJECTION III

- Alternativen
  - Factories
  - Service Locators
- aber: beide schaffen im Gegensatz zur DI neue Abhängigkeiten

UMKEHRUNG VON ABHÄNGIGKEITEN

- Aufruf-Abhängigkeiten (konkret)
  - erfordern Änderungen des Aufrufers bei neuen Aufgerufenen
  - können überführt werden in:
- Vererbungsabhängigkeiten
  - Aufrufer definiert Interface, das vom Aufgerufenen implementiert werden muss

INTERPRETATION VON INTERFACES

- Klasse: Menge gleichartiger Objekte
- kontextspezifische Interfaces: Rollen (-able, -ible)
- allgemeine Interfaces: inhaltliche Verwandtheit (vgl. abstrakte Klassen)

BLACK- UND WHITEBOX FRAMEWORKS

- Whitebox:
  - Vererbung
  - offene Rekursion
- Blackbox
  - Server/Client Interfaces
  - Open for extension, closed for modification