Grundlagen der Objektorientierung

Interfaces und abstrakte Klassen (bei Java und UML)

Stand 21.04.2015

Interfaces und abstrakte Klassen

- Elemente zur Modellierung von:
  - gemeinsamem Verhalten
  - Vererbungen / Generalisierungen / „Gruppierungen“ / Abstraktionen
- Erhöhung der Flexibilität => Wiederverwendbarkeit
- Wesentliche Elemente u.a. in Design Patterns

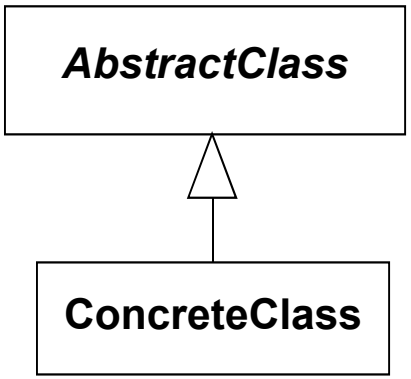
Abstrakte Klassen

- Können **nicht** instanziiert werden.
- Können implementierte **und** lediglich deklarierte Methoden enthalten.
- Dienen dazu, als Basisklasse (Oberklasse) genutzt zu werden, wobei ausschließlich die Unterklassen instanziiert werden sollen.
- Bei der Implementierung **müssen alle** nur deklarierten Methoden realisiert (implementiert) werden.

Abstrakte Klassen bei Java und UML

Java:

- Ableitung (Vererbung) mit Schlüsselwort *extends*
- Deklaration der abstrakten Methoden mit *abstract*
- Werden nicht alle abstrakten Methoden einer abstrakten Klassen in der Unterklasse implementiert, so muss diese Unterklasse ebenfalls als abstrakt deklariert werden.



UML:

- Vererbungspfeil wie bei „normalen“ Klassen
- Abstrakte Klassen werden durch einen *kursiv* geschriebenen Namen dargestellt

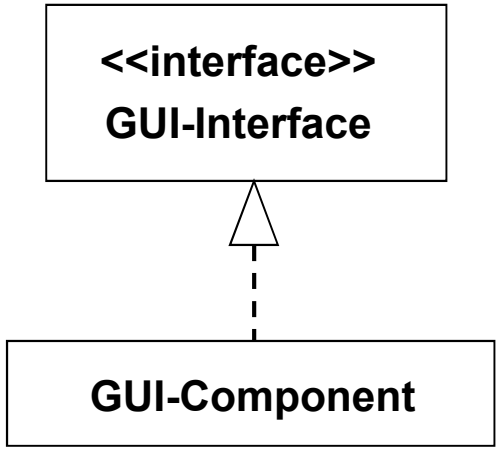
Interfaces

- Können **nicht** instanziiert werden.
- Können wie Klassen weitervererbt werden (auch Mehrfachvererbung bei Java!).
  Beispiel: *java.util.Collection* vererbt an *BeanContext*, *BeanContextServices*, *List*, *Set* und *SortedSet*
- Enthalten **ausschließlich** deklarierte Methoden und Konstanten.
  - Es können keine implementierten Methodenrümpfe existieren.
  - Es können keine dynamischen Referenzen auf andere Klassen existieren
  - Es können nur Methoden modelliert werden, die mit diesen Referenzen „arbeiten“ (z.B. get-/set-Methoden)
- Sie dienen dazu, als Template (Schablone) für die Implementierung des Verhaltens und der Eigenschaften einer „Gruppe“ von Klassen genutzt zu werden.
- Bei der Implementierung **müssen alle** Operationen realisiert (implementiert) oder durch abstrakte Klassen an deren Unterklassen „durchgereicht“ werden.
- Eine Klasse kann mehrere Interfaces implementieren.

Interfaces bei Java und UML

Java:

- Implementierung mit Schlüsselwort *implements*
- Deklaration der Methoden ohne Methodenrumpf (geschweifte Klammern) und mit Semikolon
- Bem.: in Java sind Interfaces auch Klassen!



UML:

- Interfaces werden wie Klassen auch durch ein Rechteck dargestellt
- Realisierungen (*implements*) werden mit gestrichelten Vererbungs-Pfeilen gezeichnet
- Zusätzlich wird über den Namen das Stereotyp <<interface>> geschrieben

Verwendung von Interfaces und abstrakten Klassen

1. Als „Platzhalter“ für Referenzen auf Klassen desselben „Typs“.

Beispiel (Java):

Verweis auf eine *Collection* anstatt auf einen *Vector*

➔ Spätere Änderungen der Containerklasse, welche ebenfalls *Collection* implementiert, ist ohne Auswirkung auf Quellcode der referenzierenden Klasse möglich.

2. Zur Trennung von Klassen für verschiedene Anwendungsbereiche

Beispiel:

Gemeinsame Oberklasse für RDB und ODB. Der Zugriff auf die jeweilige Datenbank ist grundlegend anders und wird in den (konkreten) Unterklassen entsprechend implementiert.

➔ z.B.Gemeinsame GUI möglich ohne oder mit minimaler Anpassung

Verwendung von Interfaces und abstrakten Klassen (2)

3. Zur Realisierung gemeinsamen Verhaltens verschiedener Klassen

Hier finden v.a. ***Interfaces*** ihre Verwendung.

Abstrakte Klassen werden dann modelliert, wenn das Verhalten der Unterklassen bereits teilweise in der Oberklasse realisiert werden kann.

4. Zur Definition programmweiter Konstanten

Beispiele:

`javax.swing.SwingConstants`,
`javax.xml.datatype.DatatypeConstants`, uvm.

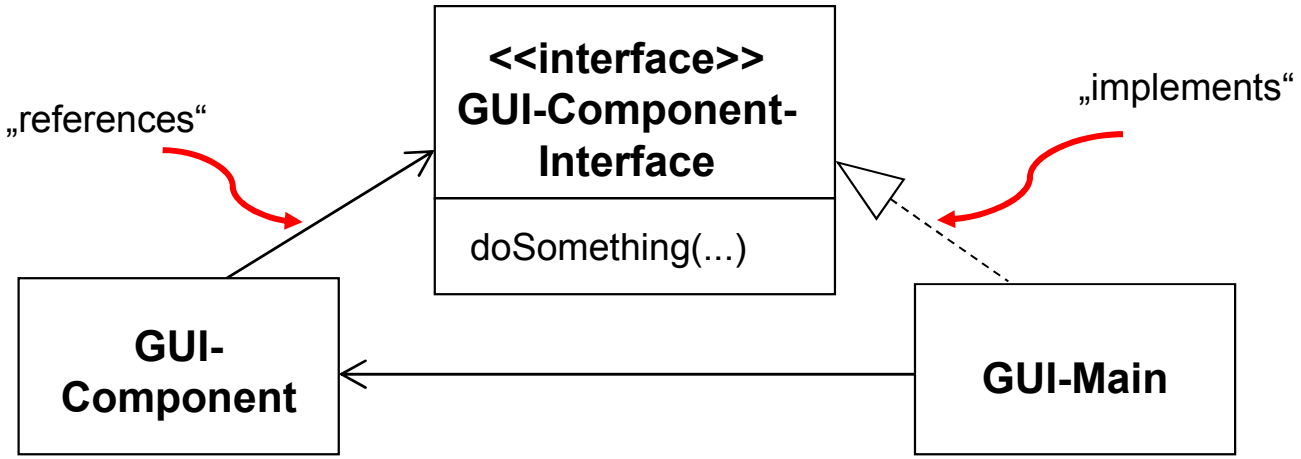
Verwendung von Interfaces und abstrakten Klassen (3)

5. Zur „Kommunikation“ zwischen zwei Klassen

Beispiel:

Eine „neutrale“ GUI-Komponente soll mit der sie erzeugenden Klasse kommunizieren, d.h. beim Drücken eines Buttons soll in dieser Klasse eine bestimmte Funktion ausgeführt werden.

➔ Deklaration dieser Funktion in einem Interface, das von der erzeugenden Klasse implementiert werden muss.



Implementierungsbeispiele

```
public abstract class ExampleAbstractClass
{
     public abstract void anyAbstractFunction( int ijk );

     public void implementedAbstractFunction( int klm ){
        System.out.println( "Der Wert von klm ist " + klm );
     }
}
```

```
public interface ExampleInterface
{
     public void anyFunction( int ijk );
     public void anyOtherFunction( int klm );
}
```

Implementierungsbeispiele (2)

Erben von einer abstrakten Klasse

```
public class ImplClass extends ExampleAbstractClass{

   // muss implementiert werden:
   public void anyAbstractFunction( int ijk ){
      // do something;
   }

   // implementedAbstractFunction() muss nicht implementiert
   // werden, da bereits in Oberklasse implementiert
}
```

Implementierungsbeispiele (3)

Einfache Implementierung eines Interfaces

```
public class ImplClass1 implements ExampleInterface {

    // alle Methoden des Interfaces müssen implementiert
    // werden:
    public void anyFunction( int ijk ) ){
        // do something;
    }
    public void anyOtherFunction( int klm ) ){
        // do something;
    }
}
```

Implementierungsbeispiele (4)

Implementierung mehrerer Interfaces

```
public class ImplClass2
          implements ExampleInterface, Runnable{

     // alle Methoden ALLER Interfaces müssen implementiert
     // werden (s.o.).
     // Probleme kann es geben, wenn zwei unterschiedliche
     // Interfaces Methoden mit dem selben Namen deklarieren!

}
```

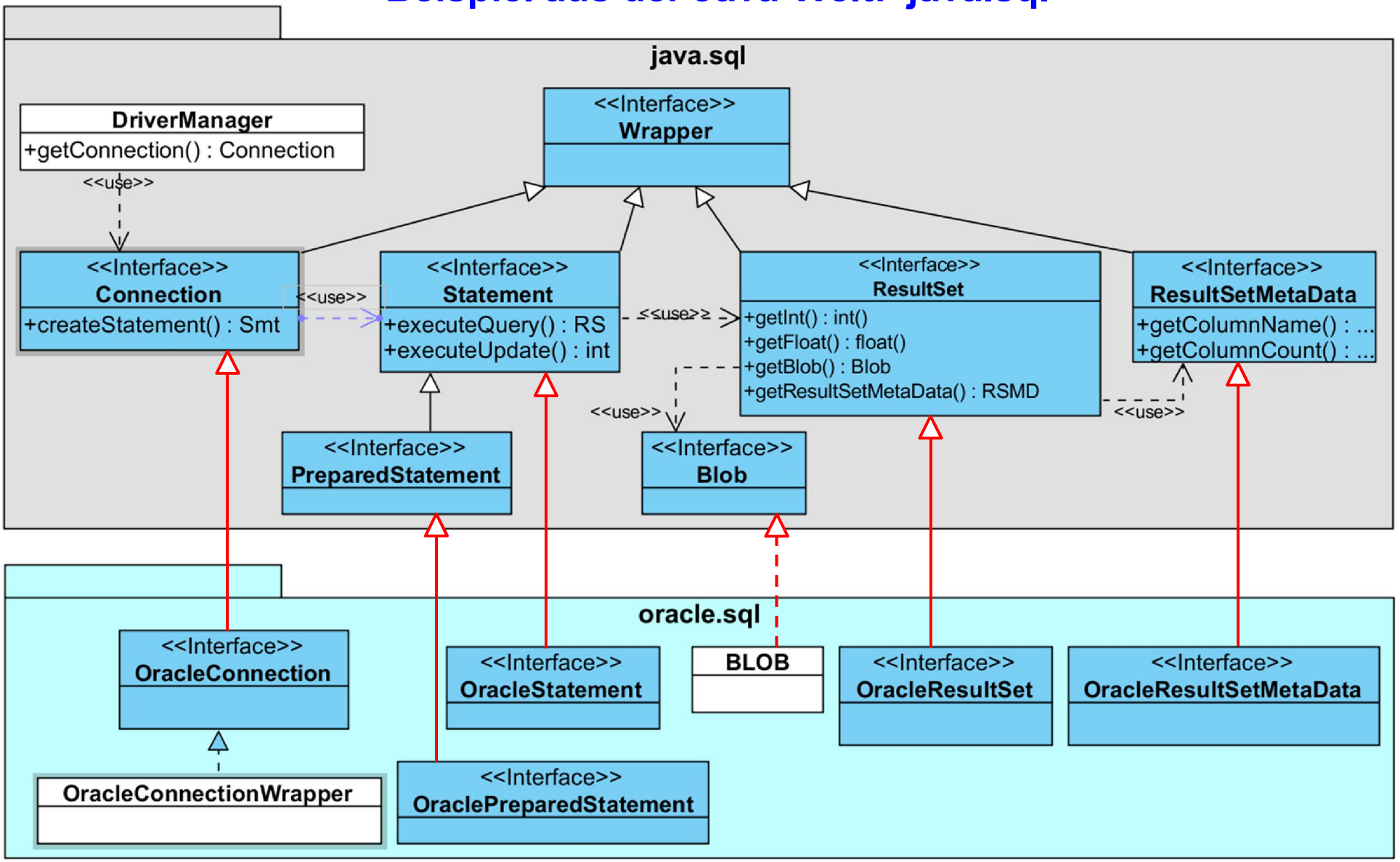
Implementierungsbeispiele (5)

Erben von einer abstrakten Klasse und gleichzeitige Implementierung eines Interfaces

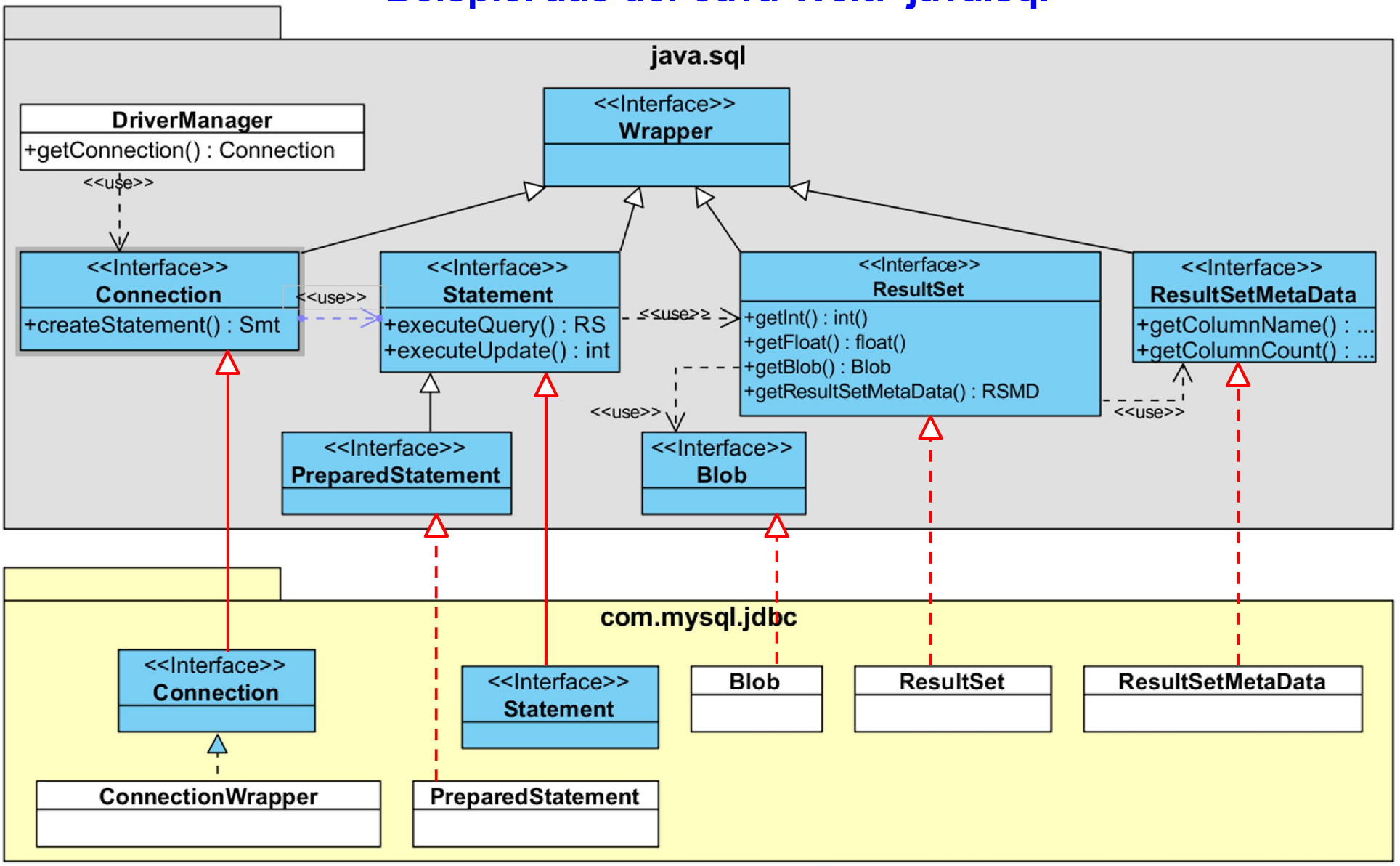
```
public class ImplClassMulti extends ExampleAbstractClass
          implements ExampleInterface {

      // alle Methoden des Interfaces sowie alle abstrakten
      // Methoden der Oberklasse müssen implementiert
      // werden (s.o.).
      // Probleme kann es geben, wenn das Interface
      // und die abstrakte Klasse zwei unterschiedliche
      // Methoden desselben Namens deklarieren!
}
```

Beispiel aus der Java-Welt: java.sql



Beispiel aus der Java-Welt: java.sql



Beispiel aus der Praxis

KIT-Projekt WISA (Wiss. Informationssystem für die Atmosphärenforschung)

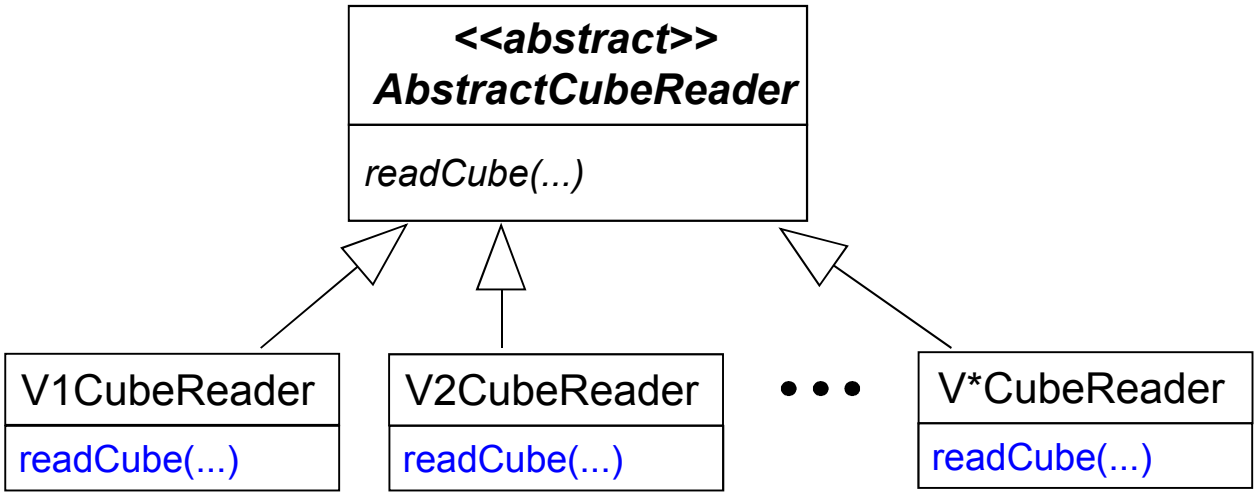
- Für die Datensätze der Instrumente MIPAS und GLORIA existieren mehrere Versionen und Plattformen (Ballon, Satellit, Flugzeug): so genannte ***Cube Files***
- Die Datensätze sind ähnlich aufgebaut, sie unterscheiden sich vor allem in den Metadaten (Versionen, Header) und dem Speichertyp (short, unsigned short, usw.)
- Alle Datensätze sollen mit einem einzigen UI-Modul visualisiert werden. Dabei sollen die einzelnen Datensätze automatisch gelesen werden, ohne dass vom Benutzer die unterschiedlichen Versionen beachtet werden müssen.
- Die erforderlichen Reader unterscheiden sich nur in wenigen Methoden (Funktionen)

Beispiel aus der Praxis

KIT-Projekt WISA (Wiss. Informationssystem für die Atmosphärenforschung)

Lösungsmöglichkeit:

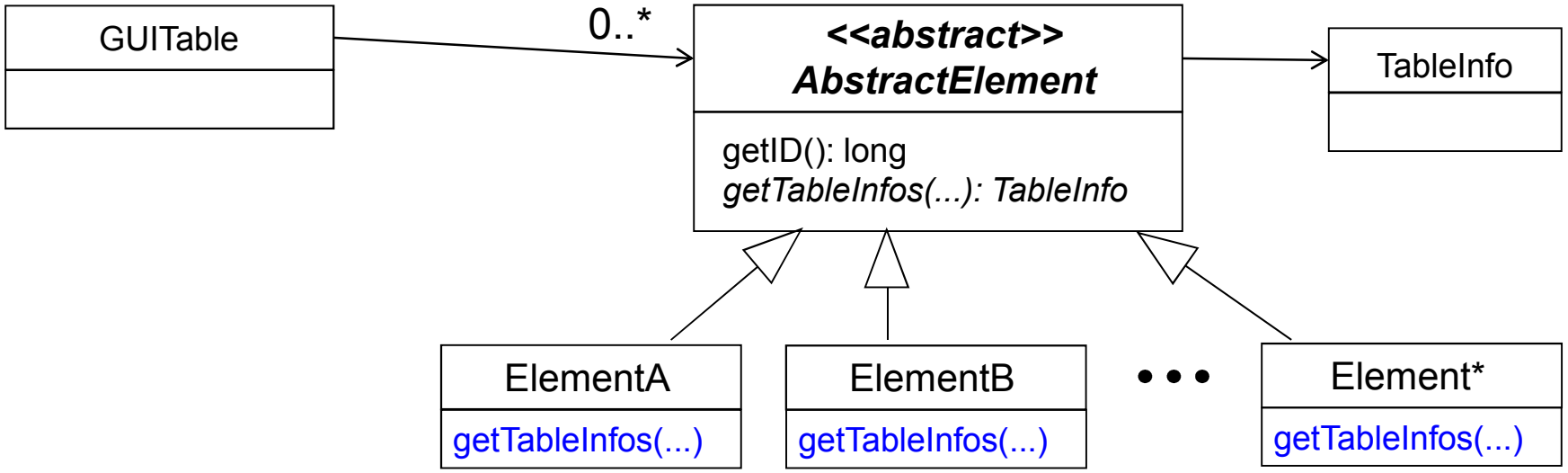
Verwendung eines (abstrakten) Readers, der die gemeinsamen Methoden realisiert sowie für jede Version (konkrete) Reader, welche die Unterschiede implementieren:



Beispiel aus der Praxis

Darstellung unterschiedlicher Klassen/Objekte in einer einfachen GUI-Tabelle

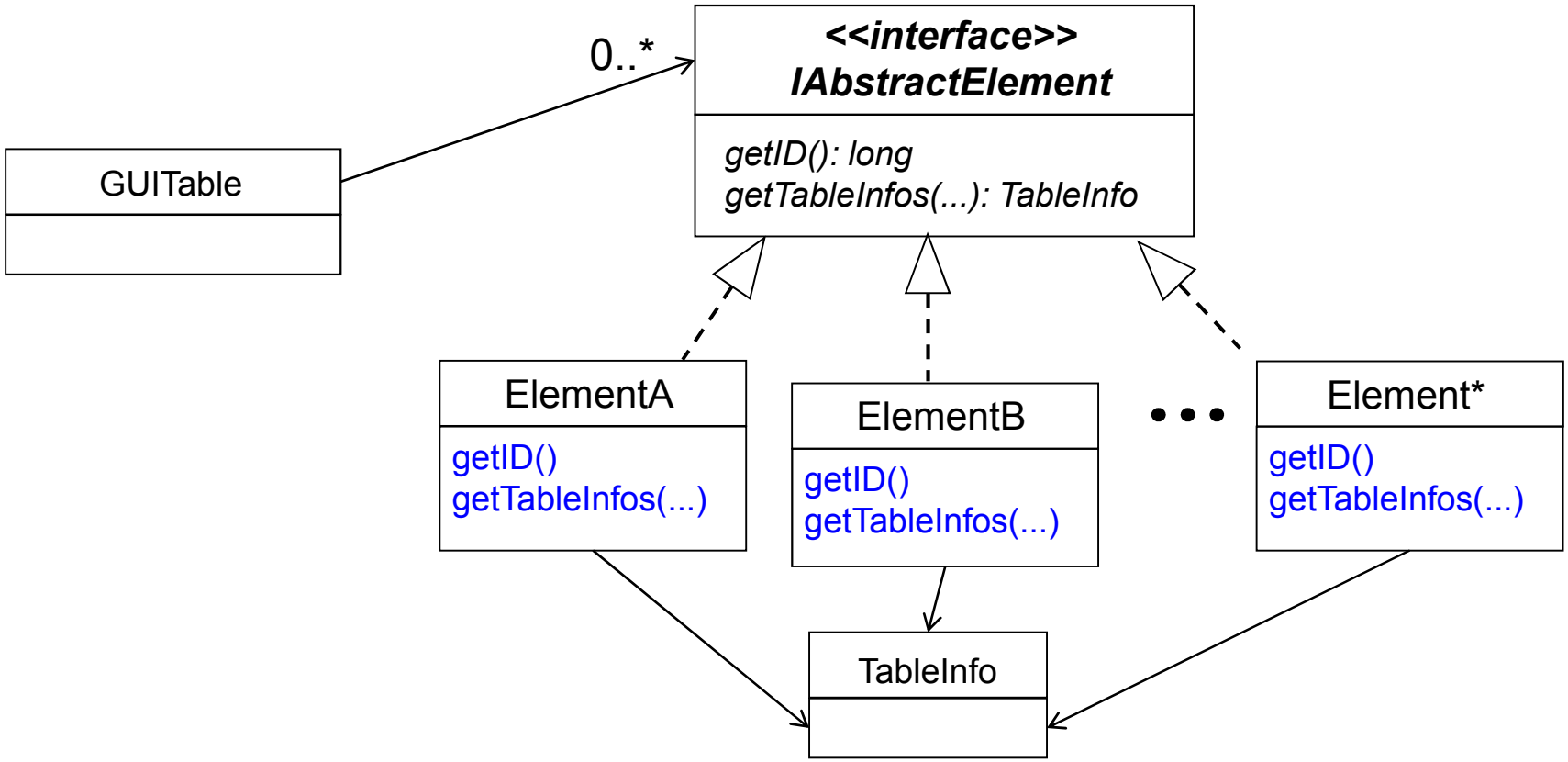
Lösungsmöglichkeit mit abstrakter Klasse



Beispiel aus der Praxis

Darstellung unterschiedlicher Klassen/Objekte in einer einfachen GUI-Tabelle

Lösungsmöglichkeit mit Interface



Beispiel aus der Praxis

Darstellung unterschiedlicher Klassen/Objekte in einer einfachen GUI-Tabelle

Lösungsmöglichkeit mit Interface UND abstrakter Klasse