Fachhochschule
Münster University of
Applied Sciences

Enhanced Collection API

Die *default* und *static* Methoden der Collection API des *JDK 1.8*

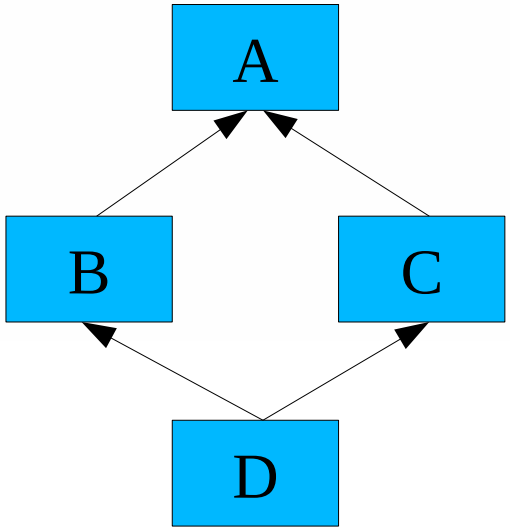
Prof. Dr. Nikolaus Wulff

Schnittstelle und Implementierung

- Die Sprache Java trennte von Anfang an eine Schnittstelle von ihrer Implementierung. Schnittstellen stellten bis zum JDK v1.8 lediglich abstrakte, public Funktionsprototypen zur Verfügung.
- ***Mehrfachvererbung*** zwischen Klassen ist in Java nicht möglich. Lediglich mehrere Interfaces können von einer Klasse implementiert werden.
- Diese Vereinfachung der Java Sprache im Vergleich zu C++ war eines der wichtigen Designziele von James Gosling dem „Vater von Java“ Anfang der 90er Jahre.

Das Rautenproblem

- Sprachen mit Mehrfachvererbung ermöglichen ein schwer zu durchschauendes Rautenproblem:



- Erbt D die Attribute und Methoden(implementierung) von A ein- oder zweimal?
- In Java ist entweder B oder C eine Schnittstelle (oder beide) und somit auch A. Die Frage stellt sich nicht...

Interface Evolution

- Die Erweiterung einer Schnittstelle um neue Eigenschaften ist als *Interface Evolution* bekannt und stellte für Java ein ernstes Problem da: Es war so gut wie unmöglich an einer öffentlichen Schnittstelle eine Änderung vorzunehmen ohne in allen implementierenden Klassen Übersetzungsfehler zu provozieren.
- Den Java Entwicklern standen die Quelltexte der implementierenden Klassen nicht zur Verfügung und sie konnten daher die Schnittstellen nicht mehr modifizieren, nachdem diese einmal veröffentlicht und in anderen Anwendungen verwendet wurde.

Wozu default Methoden?

- Eine veröffentlichte Java Schnittstelle kann nachträglich schwer oder gar nicht mehr verändert werden.
- Dies fällt besonders bei den Collection Klassen auf, sie sollten um eine ***forEach*** Schleife, die Lambda Ausdrücke verwendet, erweitert werden.
- Die ***forEach*** Methode sollte zentral, weit oben in der Vererbungshierarchie verankert werden, um in allen Collections zur Verfügung zu stehen.
- Dies hätte bedeutet alle Implementierungen hätten angepasst werden müssen. Nur den Entwicklern waren diese teilweise überhaupt nicht bekannt...

Default Methoden

- JDK 1.8 stellt per ***default*** Schlüsselwort direkt eine Implementierung innerhalb der Schnittstelle bereit.
  - Java Schnittstellen entwickeln sich hierdurch in Richtung einer Abstrakten Klasse ohne Attribute.
- Abgeleitete Klassen sind nun nicht gezwungen diese zu implementieren, können dies aber, um einen besseren Algorithmus bereit zu stellen.
- Die Collection Klassen des JDK1.8 machen regen Gebrauch von ***default*** Methoden, für die ***forEach*** Schleife und auch für Sortier Methoden.

java.lang.Iteratable

- Alle Java Collection Schnittstellen erben von *Iteratable* eine **default *forEach*** Implementierung:

```
public interface Iterable<T> {

    Iterator<T> iterator();

    default void forEach(Consumer<? super T> action) {

        Objects.requireNonNull(action);

        for (T t:this)

            action.accept(t);

    }

}
```

Default Implementierung

- Die Schnittstelle I_1 stellt eine *foo* Methode mit **default** Implementierung zur Verfügung.

```
public interface I1 {

    default void foo() {
        System.out.printf("%s using %s \n",
                this.getClass().toString(),
                I1.class.toString());
    }

}
```

- Klassen die I_1 implementieren haben automatisch eine fertige *foo* Methode, können diese aber auch überladen.

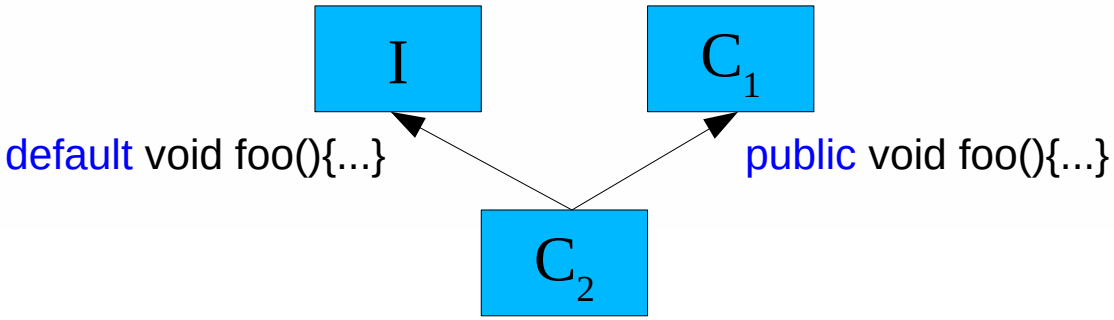
Das Rautenproblem

Implementiert eine Klassen mehrere Schnittstellen stellt sich nun auch in Java die Frage nach dem Rautenproblem:

- Das Rautenproblem für die Attribute ergibt sich nicht, da Java Schnittstellen nach wie vor keine nicht finalen nicht statischen Membervariablen erlauben.
- Auf Seite der Methoden sind nun allerdings neue Regeln notwendig, da hier nach wie vor eine Raute möglich ist, die unter Umständen zu Komplikationen führt.

Fall 1

- Superklasse und Schnittstelle implementieren dieselbe Funktion:

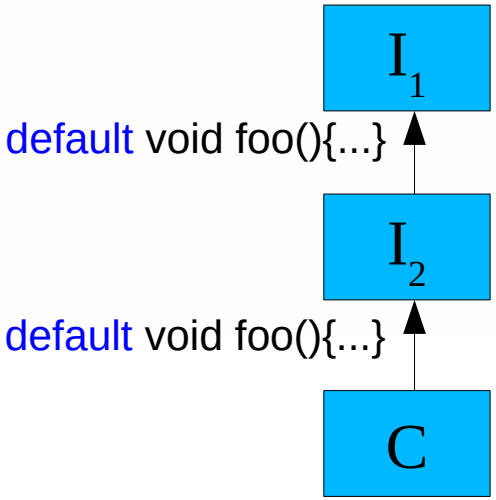


Regel:

- Die *foo* Implementierung der Superklasse C_1 wird an C_2 vererbt, die Implementierung aus I wird ignoriert.

Fall 2

- Zwei Schnittstellen implementieren dieselbe Funktion:

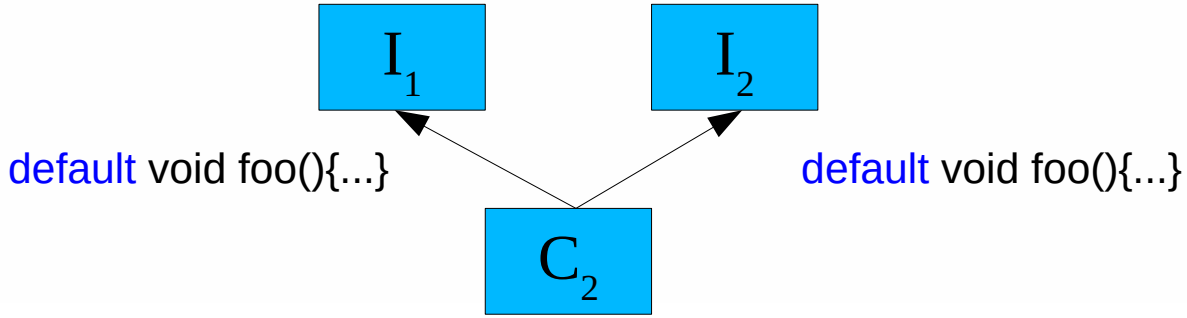


Regel:

- Die *foo* Implementierung von I_2 wird an C vererbt.

Fall 3

- Zwei Schnittstellen implementieren dieselbe Funktion ohne Vererbungsbeziehung:



- Diese Kombination ist nicht eindeutig und wird einen Übersetzungsfehler hervorrufen!
- Um den Konflikt aufzulösen muss explizit die zu verwendende *foo* Schnittstellenimplementierung per super Schlüsselwort angegeben werden.

Konfliktauflösung für Fall 3

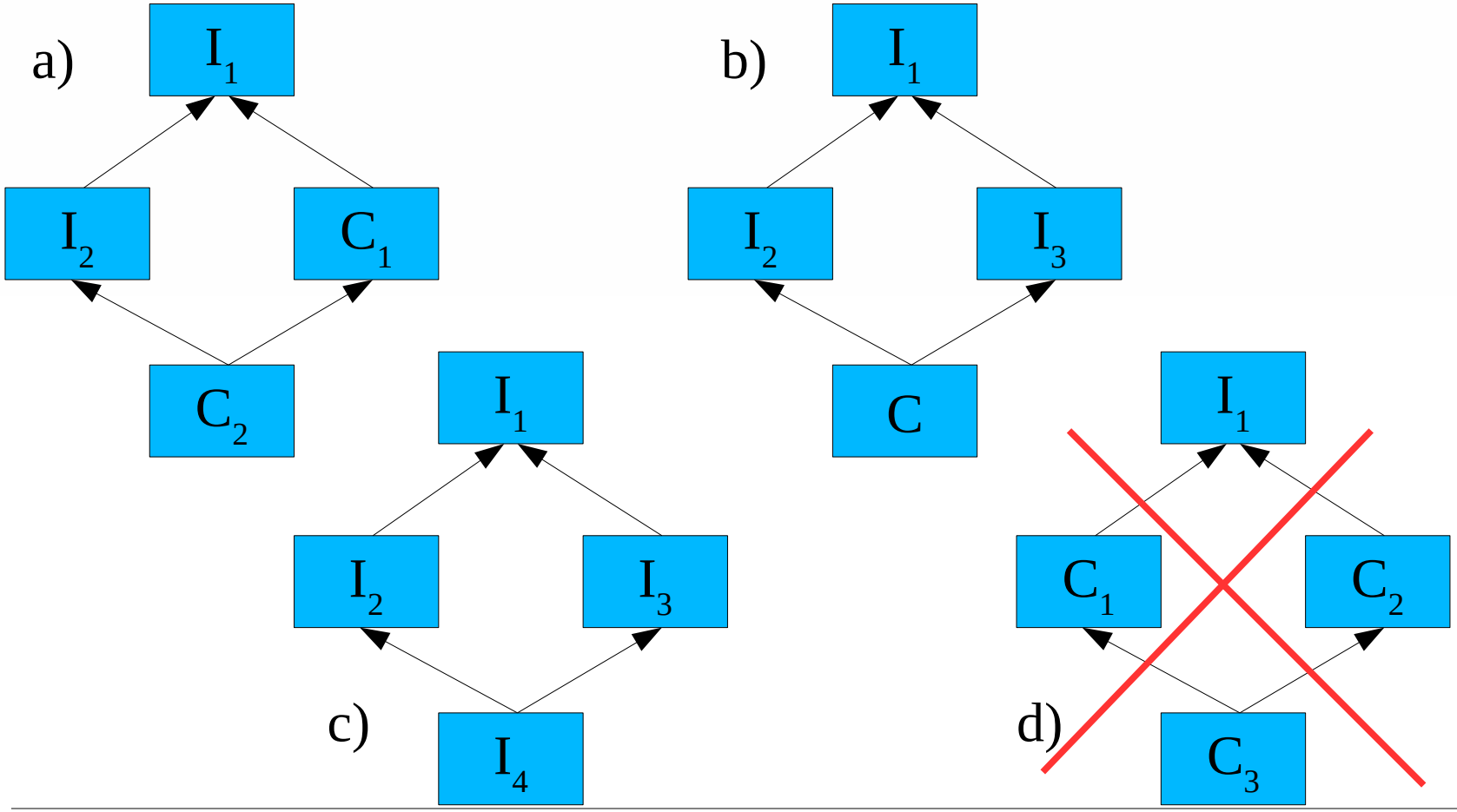
```
public class C3 implements I1, I2 {
    @Override
    public void foo() {
        I2.super.foo();
    }
```

Regel:

- Besitzen sowohl I1 als auch I2 eine **default** *foo* Methode muss der Programmierer explizit eine eigene *foo* Implementierung angeben und kann an die gewünschte **default** Implementierung einer der Schnittstellen per super Schlüsselwort delegieren.

Weitere Fälle

- Aus den Grundmustern 1 – 3 lassen sich mögliche Rauten kombinieren:



Rauten

- Die Fälle 4 a) – c) lassen sich auf die Fälle 1) – 3) zurückführen. Fall 4 d) scheidet von vornherein aus wegen der verbotenen Klassenmehrfachvererbung .
- a) I_2 überlädt I_1 wird jedoch durch C_1 ersetzt.
- b) I_2 und I_3 überladen jeweils I_1. C muss explizit foo implementieren, kann aber an I_2 oder I_3 delegieren – muss dies aber nicht.
- c) Für I_4 gilt dasselbe wie im Fall b) wobei I_4 die *foo* Methode wieder als abstract und nicht implementiert (re)deklarieren kann oder aber neu implementiert.

Generische default Methoden

- Default Methoden lassen sich auch für generische Schnittstellen definieren, wodurch weiter Ausdrucks-möglichkeiten entstehen.

```
public interface Adder<T extends Number> {

    T create(double x);

    default <A extends Number, B extends Number> T add(A a, B b) {
        return create(a.doubleValue() + b.doubleValue());
    }
}
```

- Die Schnittstelle *Adder* verbindet eine default Methode mit einer generischen Fabrikmethode zur Erzeugung von T Instanzen...

Verketten von Schnittstellen

- Mittels **default** Methoden lassen sich effektiv Operationen verketten:

```
public interface Operation<T> {

    T op(T a, T b);

    default Operation<T> andThen(Operation<T> after) {
        return (T a, T b) -> after.op(this.op(a, b),b);
    }

}
```

- Eine neue Operation Instanz wird per *andThen* Aufruf durch Verketten von *this* und *after* erzeugt.

Verkettungsbeispiel

- Zwei Operationen werden durch λ-Ausdrücke implementiert und per *andThen* eine weitere Operation als Verkettung erstellt:

```
Operation<Long> add = (Long a, Long b) -> a + b;
Operation<Long> mul = (Long a, Long b) -> a * b;

Operation<Long> mix = mul.andThen(add);

long x = 3, y = 2, z;

z = mix.op(x, y);
```

- Solche Verkettungen kommen bei vielen Schnittstellen des *java.util.function* Pakets vor...

Statische Methoden

- Zusätzlich zu default Methoden lassen sich nun auch statische Methoden per static Schlüsselwort direkt in einer Schnittstelle implementieren.
- Diese darf nur direkt über den Schnittstellennamen angesprochen werden und wird nicht vererbt.

```
public interface Operation<T> {
    T op(T a, T b);
    default Operation<T> andThen(Operation<T> after) {
        return (a,b) -> after.op(this.op(a, b), b);
    }

    static <C extends Comparable<? super C>> Operation<C> min() {
        return (a,b)-> a.compareTo(b)<0 ? a : b;
    }
}
```

Statische Methode

Zusammenfassung

- Java wurde um die Fähigkeit der Implementierung von Funktionen in einer Schnittstelle erweitert.
- Es gibt default und static Schnittstellen Methoden.
- Das Prinzip der Mehrfachvererbung – mit all seinen Vor- und Nachteilen – ist in Java realisiert.
- Das Rautenproblem muss seit dem JDK 1.8 in Java neu betrachtet werden.
- Lambda Ausdrücke und Generics sind mit den neuen Sprachmöglichkeiten besser in Java integriert und Schnittstellen können abwärtskompatibel erweitert werden ohne bestehenden Code zu brechen.